2010

Learning From Geometry In Learning For Tactical And Strategic Decision Domains

Jason Gauci
University of Central Florida

Learning From Geometry in Learning for Tactical and Strategic Decision Domains

by

Jason Gauci
M.S. University of Central Florida, 2006
B.S. University of Central Florida, 2004

A dissertation submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy
in the Department of Electrical Engineering and Computer Science
in the College of Engineering and Computer Science
at the University of Central Florida
Orlando, Florida

Fall Term
2010

Major Professor:
Kenneth O. Stanley

Abstract

Artificial neural networks (ANNs) are an abstraction of the low-level architecture of biological brains that are often applied in general problem solving and function approximation. Neuroevolution (NE), i.e. the evolution of ANNs, has proven effective at solving problems in a variety of domains. Information from the domain is input to the ANN, which outputs its desired actions. This dissertation presents a new NE algorithm called Hypercube-based NeuroEvolution of Augmenting Topologies (HyperNEAT), based on a novel indirect encoding of ANNs. The key insight in HyperNEAT is to make the algorithm *aware* of the geometry in which the ANNs are embedded and thereby exploit such domain geometry to evolve ANNs more effectively. The dissertation focuses on applying HyperNEAT to tactical and strategic decision domains. These domains involve simultaneously considering short-term tactics while also balancing long-term strategies. Board games such as checkers and Go are canonical examples of such domains; however, they also include real-time strategy games and military scenarios. The dissertation details three proposed extensions to HyperNEAT designed to work in tactical and strategic decision domains. The first is an action selector ANN architecture that allows the ANN to indicate its judgements on every possible action all at once. The second technique is called substrate extrapolation. It allows learning basic concepts at a low resolution, and then increasing the resolution to learn more advanced concepts. The

final extension is geometric game-tree pruning, whereby HyperNEAT can endow the ANN the ability to focus on specific areas of a domain (such as a checkers board) that deserve more inspection. The culminating contribution is to demonstrate the ability of HyperNEAT with these extensions to play Go, a most challenging game for artificial intelligence, by combining HyperNEAT with UCT.

To my wife, Lindsey and to my parents, John and Nadia.

Acknowledgments

Thanks to my advisor Dr. Kenneth O. Stanley, who introduced me to neuroevolution and mentored my development through the Ph.D. program. Much of this dissertation exists solely because of his creative genius and attention to detail.

Thanks to my dissertation committee members Dr. Gita Sukthankar, Dr. Michael Georgiopoulos, and Dr. Paul Wiegand for offering their advice and donating their time and suggestions.

I'd like to thank my wife, who has been a constant source of inspiration and guidance through the Ph.D. process. I would like to thank my mother for reading to me and taking me to the library as a child. I'd also like to thank my father, who continuously worked hard to provide me with the education I needed to succeed. I hope that this dissertation can be a symbol of appreciation for their dedication and investment in my future.

Thanks to past and present members of the Evolutionary Complexity Research Group (Eplex) at University of Central Florida (http://eplex.cs.ucf.edu) for their valuable feedback on my various papers, projects, and presentations. Namely, Dr. Charles E. Bailey, David D'Ambrosio, Dr. Erin Hastings, Amy Hoover, Joel Lehman, Sebastian Risi, and Phillip Verbancsics.

Special thanks to those who have downloaded the HyperNEAT C++ source code and contributed back to the project with bug fixes and published results, namely Jeff Clune, Evert Haasdijk, and Krzysztof Krawiec.

Special thanks to Sébastien Fourey for his libBoard library, which created the substrate images rendered in this dissertation. Thanks to Martin Fierz for his Cliché and Cake checkers engines, and also to Markus Enzenberger, Martin Müller and Broderick Arneson for the Fuego Go engine and player.

HyperNEAT C++ was created with Boost, WxWidgets, Python, OpenMPI, Zlib, TinyXML, and PyOpenGL. Thanks to all of the developers of these libraries, and to all open source developers, who tirelessly donate their time and resources to promote free and open source software around the world.

Jason Gauci

University of Central Florida

November 2010

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

This dissertation introduces a new method for training artificial neural networks (ANNs) in tactical and strategic decision-making domains. These domains typically involve two mechanics: (1) searching through a game-tree of potential future states and (2) evaluating the quality of such states. Importantly, knowledge about the world is often geometric. For example, when a new player learns a new tactic in chess, e.g. forking two pieces or pinning a piece to the king, it is often necessary to extrapolate the technique to novel board positions. This extrapolation requires an understanding of the mobility of pieces with respect to the geometry of the board. Such understanding means appreciating *regularities* i.e. situations with equivalent implications that can occur at different positions within the board geometry.

Current methods that evolve ANNs often represent neural structure directly. That is, each node and link in the ANN, also called the *phenotype*, is mapped one-to-one from a gene in the *genotype* [SM02, FFP90, GM97, Yao99, FDM08]. Because of this direct mapping, the complexity of the genotype (and therefore the difficulty of learning) is often directly correlated with the size of the phenotype being evolved. Because this direct mapping creates a need for smaller phenotypes, current methods often require human engineering to compress the state of the world into a small set of meaningful inputs that the neural network can

process. In effect, such a process requires human engineering to convert sensory experience into a vector of floating point numbers. Once encoded into a vector, however, the system cannot directly perceive geometric relationships among the input variables. Human engineering is required to exploit any geometric regularities as part of the input encoding process. For example, ANNs that classify images are sometimes provided inputs from a *feature-space* that encodes regularities in the image because the ANNs themselves cannot perceive such regularities [Fra92, GU92, HL01].

In fact, geometry is integral to extracting such underlying features. If the network could somehow *see* the domain geometry, it might be possible for it to learn from raw inputs without the need for a human to include higher-level features. Arguably, *automatically* abstracting from raw inputs to feature space is a step towards higher-level machine learning. Accordingly, the main hypothesis of this paper is that automatically learning from geometry benefits machine learning in tactical and strategic decision domains. Although they also have a geometric interpretation, convolutional neural networks [LBH98] and support vector machines [HDO02] do not naturally apply in such domains because there is no explicit error signal, which is why human engineering in tactical and strategic tasks has been popular. In contrast, this dissertation focuses on the ability of machines to learn from geometry in tactical and strategic domains without human engineering. A method to learn from geometry that I co-invented is introduced and new capabilities that result from this method are explored.

The method for learning from geometry is *Hypercube-based NeuroEvolution of Augmenting Topologies* (HyperNEAT) [GS08a, SDG09, GS10a]. HyperNEAT evolves ANNs that are

constructed with an awareness of their relationship to the problem geometry. The key to learning from geometry in HyperNEAT is an *encoding* called *compositional pattern producing networks* (CPPNs). A CPPN takes geometry as input and outputs patterns of connection weights that are a function of geometric space. In this way, CPPNs encode ANNs. HyperNEAT's capabilities are demonstrated in visual discrimination and tactical and strategic decision-making domains. Because HyperNEAT excels in these domains, it is possible to leverage its existing capabilities to investigate new directions that have so far eluded machine learning.

In addition to HyperNEAT, three extensions are presented and HyperNEAT, with these extensions, is shown to be effective in the challenging domain of computer Go. First, an *action-selector* ANN architecture is introduced that allows the ANN to indicate its desired move directly instead of evaluating the value of possible future states. Next, the unique scaling properties of CPPNs allow a new kind of incremental evolution called *substrate extrapolation*, whereby the CPPNs evolve small-scale ANNs during the initial generations of evolution, after which the ANNs increase in scale to allow more complex solutions to evolve. Finally, HyperNEAT's ability to create large-scale ANNs allows a new kind of game-tree constraint called *geometric game-tree pruning*, wherein HyperNEAT evolves an ANN that dictates what areas of the game-tree deserve focus, and biases the game-tree search based on this focus.

After these extensions are introduced, Go is presented as a killer application for HyperNEAT. Go is challenging for artificial intelligence but HyperNEAT's ability to learn from

geometry and construct patterns of ANN weights make HyperNEAT particularly suited for the task of playing Go. Substrate extrapolation allows HyperNEAT to learn Go on a smaller board and then extrapolate this knowledge to play on larger boards, a technique that is unprecedented in neuroevolution for ANNs that take the entire board as simultaneous input. Currently, the most effective computer Go players employ an algorithm named *Upper Confidence bounds applied to Trees* (UCT). UCT currently relies on statistical information and human engineering to bootstrap the learning process [GS07b]; however, HyperNEAT with geometric game-tree pruning can replace such human engineering, advising UCT more effectively and resulting in an even stronger Go player.

1.1 CONTRIBUTIONS

In summary, this dissertation contributes a major new neuroevolution algorithm, along with several enhancements from tactical and strategic decision domains, and applies these enhancements to a popular domain:

1. This dissertation is the first to introduce HyperNEAT, an indirect encoding that I co-invented with Kenneth O. Stanley and David D'Ambrosio.

2. An extension of HyperNEAT called *substrate extrapolation* is introduced, which creates the capability to generate ANNs that can scale with the domain.

3. The first *action-selector* is evolved for playing the tactical and strategic game of Go. The action selector is an ANN containing thousands of connections that can specify where it wants to move directly (instead of returning a traditional board evaluation), which has never before been attempted by neuroevolution.

4. A new idea called *geometric game-tree pruning* extends to action-selection by allowing HyperNEAT to prune a search tree that is traversed by a search algorithm. HyperNEAT-Cake, a combination of HyperNEAT and the Cake checkers engine, demonstrates HyperNEAT's ability to extend search algorithms in this way effectively.

5. Finally, HyperNEAT-UCT, combines of HyperNEAT and UCT, an effective search algorithm for Go. Adding HyperNEAT gives UCT the ability to bootstrap with prior knowledge from previous experiences, and is shown to be more effective than beginning with no prior knowledge.

1.2 OUTLINE

The dissertation begins with background on neuroevolution and CPPNs. Chapter 3 then explains the new HyperNEAT method that generates ANNs with connective CPPNs. Chapter 4 presents results on HyperNEAT in the strategy game of checkers. Chapter 5 introduces continuous and discrete substrate extrapolation, and Chapters 6 and 7 demonstrate both extrapolation techniques with two case studies. Chapter 8 demonstrates the geometric game-

tree pruning extension to HyperNEAT. Chapter 9 then explores Go as a killer application for HyperNEAT combined with UCT search. Finally, Chapter 10 discusses the aforementioned topics and the implications of their results in depth.

CHAPTER 2

BACKGROUND

This chapter first reviews the fields of neuroevolution and generative and developmental systems, then provides an overview of CPPNs, which are capable of generating complex spatial patterns in Cartesian space, and describes the NEAT method that evolves them. It concludes with a discussion of the role of geometry in machine learning.

2.1 NEUROEVOLUTION

Neuroevolution (NE) is an area of evolutionary computation that focuses on training neural networks through evolutionary algorithms [FDM08, Yao99]. This approach applies the concepts of fitness, generation, populations, mutation, and crossover from evolutionary algorithms to evolve ANNs. It also benefits from the neural model, which is based on biology. In NE, the *genotype* represents an individual in the genetic algorithm that is transformed into a *phenotype* during evaluation that is an ANN. After evaluation, the genotype receives a *fitness* that decides the parents of the next generation of individuals. NE can evolve any kind of ANN, including recurrent and adaptive networks [SBM08a, RS10]. The way that

the phenotype is described by a genotype is called the *encoding* of that phenotype. This dissertation focuses on a new encoding that leverages geometry to create regular phenotypes.

In early NE research, humans dictated the topology of evolved ANNs [IJC89, SP91]. While this approach allows human experts to design the topology with domain-specific optimizations, the approach is also limited by its fixed topology. In contrast, evolving structure in addition to connection weights removes the burden of deciding the network topology from humans and places it on the learning algorithm [ASP93, KR91, SM02, Gru95].

The first methods to evolve both network structure and connection weights encoded networks directly, which means that a single gene in the genotype maps to a single connection in the phenotype [Yao99]. While this approach is straightforward, the problem is that it requires learning each connection weight individually. As a result, it is impossible to learn a regular pattern of connectivity without learning each connection in the pattern on its own. Again, human engineering is one approach to overcoming this limitation. For example, Togelius and Lucas [TL05] introduced a symmetric ANN to power a symmetric robot, which reduced the amount of evaluations required by a factor of eight. Chellapilla and Fogel [CF99] created an ANN that was capable of expert level checkers play by engineering an ANN with separate processing nodes for each subsquare of the board. Human engineering can capture patterns and regularities in the input and reduce them to a vector of numbers. However, ideally, evolution should be able to capture patterns and regularities on its own.

Indirect encodings give evolution the opportunity to explore patterns and regularities by encoding the genotype as a *description* that maps indirectly to the phenotype [GWP96,

Kit90, SM03]. That way, the genotype can be much smaller than the phenotype, which results in fewer variables to optimize for the evolutionary algorithm. In fact, the CPPNs in HyperNEAT are a specialized kind of indirect encoding that draws inspiration from biology. The next section describes the field that studies such biologically-motivated indirect encodings.

2.2 GENERATIVE AND DEVELOPMENTAL SYSTEMS (GDS)

In biological genetic encoding the mapping between genotype and phenotype is indirect. The phenotype typically contains orders of magnitude more structural components than the genotype contains genes. Thus the only way to discover such high complexity may be through a mapping between genotype and phenotype that translates few dimensions into many, i.e. through an *indirect encoding*. A most promising area of research in indirect encoding is *generative and developmental encoding*, which is motivated from biology [Ang95, BK99, HP02, SM03, Bon02]. In biological development, DNA maps to a mature phenotype through a process of growth that builds the phenotype over time. Development facilitates the reuse of genes because the same gene can be activated at any location and any time during the development process.

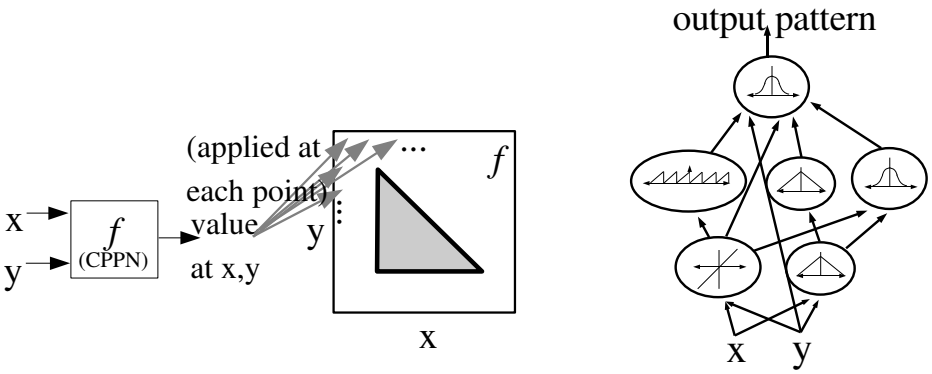
This observation has inspired an active field of research in generative and developmental systems [BK99, Bon02, Fed04, HP02, Lin74, Mil04, MSR91, SM03, Tur52]. The aim is to find

an abstraction of natural development for a computer running an evolutionary algorithm, so that evolutionary computation can begin to discover complexity on a natural scale. Prior abstractions range from low-level cell chemistry simulations to high-level grammatical rewrite systems [SM03].

2.3 COMPOSITIONAL PATTERN PRODUCING NETWORKS

This section describes an indirect encoding motivated by development based on a unique high-level abstraction. Compositional Pattern Producing Networks (CPPNs) are a novel abstraction of development that can represent sophisticated repeating patterns in Cartesian space [Sta06b, Sta07]. Unlike most generative and developmental encodings, CPPNs do not require an explicit simulation of growth or local interaction, yet still exhibit their essential features. The remainder of this section reviews CPPNs, which are augmented in HyperNEAT to represent connectivity patterns and ANNs.

Consider the phenotype as a function of n dimensions, where n is the number of dimensions in physical space. For each coordinate in that space, its level of expression is an output of the function that encodes the phenotype. Figure 2.1a shows how a two-dimensional phenotype can be generated by a function of two parameters. A mathematical abstraction of each stage of development then can be represented explicitly inside this function.

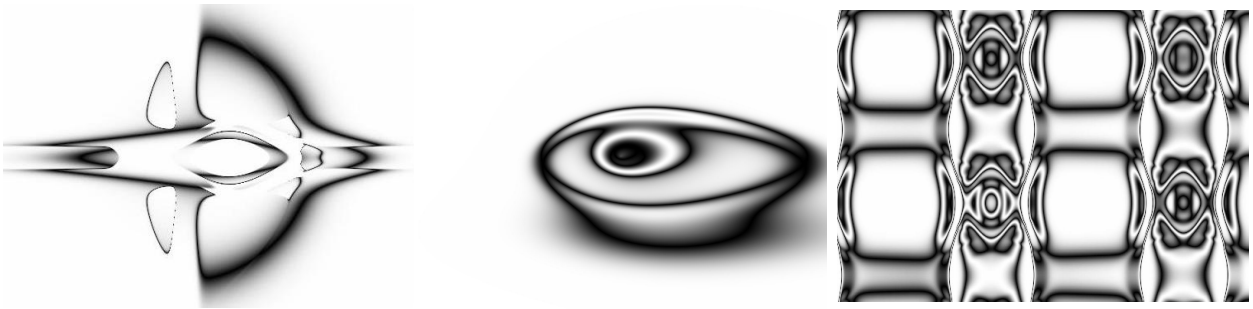


(a) Mapping (b) Composition

Figure 2.1: **CPPN Encoding**. (a) The function f takes arguments x and y, which are coordinates in a two-dimensional space. When all the coordinates are drawn with an intensity corresponding to the output of f, the result is a spatial pattern, which can be viewed as a phenotype whose genotype is f. (b) The CPPN is a graph that determines which functions are connected. The connections are weighted such that the output of a function is multiplied by the weight of its outgoing connection.

Stanley [Sta06b, Sta07] showed how simple canonical functions can be composed to create networks that produce complex regularities and symmetries. Each component function creates a novel geometric *coordinate frame* within which other functions can reside. The main idea is that these simple canonical functions are abstractions of specific events in development such as establishing bilateral symmetry (e.g. with a symmetric function such as Gaussian) or the division of the body into discrete segments (e.g. with a periodic function such as sine). Figure 2.1b shows how such a composition is represented as a network.

Such networks are called *Compositional Pattern Producing Networks* because they produce spatial patterns by composing basic functions. While CPPNs are similar to ANNs, they differ in their set of activation functions and how they are applied. Furthermore, they are an abstraction of development rather than of biological brains.



(a) Symmetry (b) Imperfect Symmetry (c) Rep. With Variation

Figure 2.2: **CPPN-generated Regularities.** Spatial patterns exhibiting (a) bilateral symmetry, (b) imperfect symmetry, and (c) repetition with variation are depicted. These patterns demonstrate that CPPNs effectively encode fundamental regularities of several different types.

Through interactive evolution, Stanley [Sta06b, Sta07] showed that CPPNs can produce spatial patterns with important geometric motifs that are expected from generative and developmental encodings and seen in nature. Among the most important such motifs are symmetry (e.g. left-right symmetries in vertebrates), imperfect symmetry (e.g. right-handedness), repetition (e.g. receptive fields in the cortex [ZBL99]), and repetition with variation (e.g. cortical columns [GC02]). Figure 2.2 shows examples of several such important motifs produced through interactive evolution of CPPNs.

These patterns are generated by applying the right activation functions (e.g. symmetric functions for symmetry; periodic functions for repetition) in the right order in the network. The order of activations is an abstraction of the unfolding process of development.

2.4 EVOLVING CPPNS

It turns out that because CPPNs are *networks*, they can be evolved by neuroevolution algorithms just like ANNs. One particularly effective such method for evolving CPPNs is NeuroEvolution of Augmenting Topologies (NEAT) because it can evolve CPPNs of *increasing complexity*, which means that the patterns they represent *also* increase in complexity. In the following sections, the NEAT method is described and CPPN-NEAT, which extends NEAT to evolve CPPNs, is then explained.

2.4.1 NEUROEVOLUTION OF AUGMENTING TOPOLOGIES

Neuroevolution of Augmenting Topologies (NEAT) is a leading method in NE [Sta03]. NEAT evolves complex ANNs by beginning with a simple topology and adding structure during evolution. NEAT handles the problem of competing conventions, and is able to protect innovation over the course of several generations [SM02]. NEAT has proven effective in several domains, including pole balancing [SM02] and robot control [SM04a]. NEAT has also performed well in several tactical and strategic decision-making tasks including Go [SM04b], squad-based combat simulation [SBM05], and Robocup Keepaway [WKM05]. This section reviews the NEAT method. Comprehensive introductions are available in Stanley and Miikkulainen [SM02] and Stanley and Miikkulainen [SM04a].

NEAT is based on three key ideas. First, to allow network structures to increase in complexity over generations, a method is needed to keep track of which gene is which. Otherwise, it is not clear in later generations which individual is compatible with which, or how their genes should be combined to produce offspring. NEAT solves this problem by assigning a unique *historical marking* to every new piece of network structure that appears through a structural mutation. The historical marking is a number assigned to each gene corresponding to its order of appearance over the course of evolution. The numbers are inherited during crossover unchanged, and allow NEAT to perform crossover without the need for expensive topological analysis. That way, genomes of different organizations and sizes stay compatible throughout evolution.

Second, NEAT speciates the population, protecting innovative structures from immediate extinction. When the population is speciated, individuals compete primarily within their own niches instead of with the population at large. This way, topological innovations are protected in a new niche where they have time to optimize their structure before competing with other niches in the population. NEAT uses the historical markings on genes to determine to which species different individuals belong. The reproduction mechanism for NEAT is *explicit fitness sharing* [GR87], in which organisms in the same species must share the fitness of their niche, preventing any one species from taking over the population.

Third, NEAT begins with a uniform population of simple networks with no hidden nodes, differing only in their initial random weights. Speciation protects new innovations, allowing diverse topologies to gradually complexify over evolution. Thus, NEAT can start minimally,

and grow the necessary structure over generations. A similar process of gradually adding new genes has been confirmed in natural evolution [Mar99, WHR87] and shown to improve adaptation [Alt94]. Through complexification, high-level features can be established early in evolution and then elaborated and refined as new genes are added [Mar99].

2.4.2 CPPN-NEAT

CPPNs are networks, so it follows that NEAT can evolve increasingly complex CPPNs. The main difference between CPPNs and ANNs is that CPPNs have several possible activation functions at each node, whereas ANN neurons usually all contain a sigmoid function. CPPN-NEAT addresses this difference by encoding the activation function in each node. When CPPN-NEAT adds a node to a CPPN, it chooses a random activation function for that node. In this way, the activation functions of nodes are evolved along with the link weights.

The CPPN-NEAT method evolves increasingly complex CPPNs that encode complex pattern such as those shown in Figure 2.2. CPPN-generated patterns evolved with NEAT exhibit several essential motifs and properties of natural phenotypes [Sta06a, Sta07]. If such properties were transferred from spatial patterns such as those in Section 2.3 to evolved *connectivity patterns*, the representational power of CPPNs could potentially evolve large-scale ANNs, as explained in Chapter 3.

2.5 ROLE OF GEOMETRY IN MACHINE LEARNING

Among the primary goals of any approach to machine learning is generalization. The ability to represent and thereby discover regularities in the *geometry* of the task domain is essential to generalization. For example, knowing the relative positions of squares in a board game is fundamental to mastering the mechanics of the game. Understanding the implications of adjacency requires recognizing the same adjacency relationships between any two squares on the board. A general understanding of board geometry makes it possible to learn general tactics rather than specific actions tied to a specific position. This central role of geometric regularity to generalization extends beyond board games to robot control, in which events at different relative positions often require similar responses, and computer vision tasks, in which the same object may appear at different positions and orientations in the retina.

To appreciate how essential geometry is to learning, imagine learning to play checkers on a board whose squares are each torn from the board and scattered across the living room randomly. The rules are the same and each square still represents the same position on the board as usual. The only problem is that the adjacency relationships among the pieces become entirely opaque to the player. Interestingly, when a board state is input into a machine learning algorithm as a flat vector of position values, the geometry of the board is no less opaque to the learner than in this satirical scenario.

Recognizing that task geometry plays a critical role in many machine learning domains, researchers have introduced a variety of methods ranging from tile coding [SB98] to special-

ized neural network topologies [CF01, TL05] that exploit geometric relationships in different ways. However, such approaches typically require the user to specify *a priori* how different regions of the task domain should be broken apart or assorted, which means the learner cannot itself discover the most essential regularities and relationships. The next section describes two such prior methods for exploiting geometry.

2.6 PRIOR WORK IN EXPLOITING GEOMETRY IN MACHINE LEARNING

This dissertation introduces HyperNEAT, which allows the learner to exploit domain geometry. However, HyperNEAT is not the first approach to incorporate geometric knowledge. This section reviews two techniques that allow the user to convey some geometric information about the domain to the learning algorithm. In contrast, the main challenge that HyperNEAT addresses is how to allow the learning algorithm *itself* to discover and exploit regularities in the geometry.

Tile coding is a common reinforcement learning technique that partitions the state space of a task into small (often overlapping) chunks. Because the state space is often geometric, e.g. in maze navigation [SS05], the partitions separate different geometric locations. By breaking the geometry into parts, each part can be learned separately as a simple subtask. While advantageous in several problem domains, a downside is that because tile coding breaks the geometry into pieces, it prevents the learner from discovering patterns and regularities

that vary across whole dimensions of the geometry. Leffler et al. [LLE07] show how this problem can be alleviated by *a priori* specifying to the learning method which tiles are related, thereby conveying useful regularities. However, they note that an ideal approach would exploit geometric regularities autonomously.

An interesting attempt to integrate geometry into evolutionary computation is Blondie24, an evolved checkers-playing artificial neural network (ANN) [CF01]. The main idea in Blondie24 is that the ANN topology can be better *engineered* to respect the regularities inherent in the game. In particular, the weights of an ANN topology engineered by hand are evolved. Every subsquare (i.e. set of positions arranged in a square shape) of the board is input to a separate hidden node responsible for only that subsquare (figure 2.3). Connections are specified from the actual board inputs to their respective subsquares, and also between the inputs and the final output node. The main idea in this engineered structure is that independent local relationships within each subsquare can be learned separately and then combined at a higher level in the network. Through coevolution (i.e. candidates were evaluated by playing against each other), Blondie24 was able to reach expert-level play on a popular Internet checkers server [CF01]. However, as with reinforcement learning, an ideal approach would remove the need for engineering by learning geometric regularities on its own. The next chapter introduces HyperNEAT, which aims to make that possible.

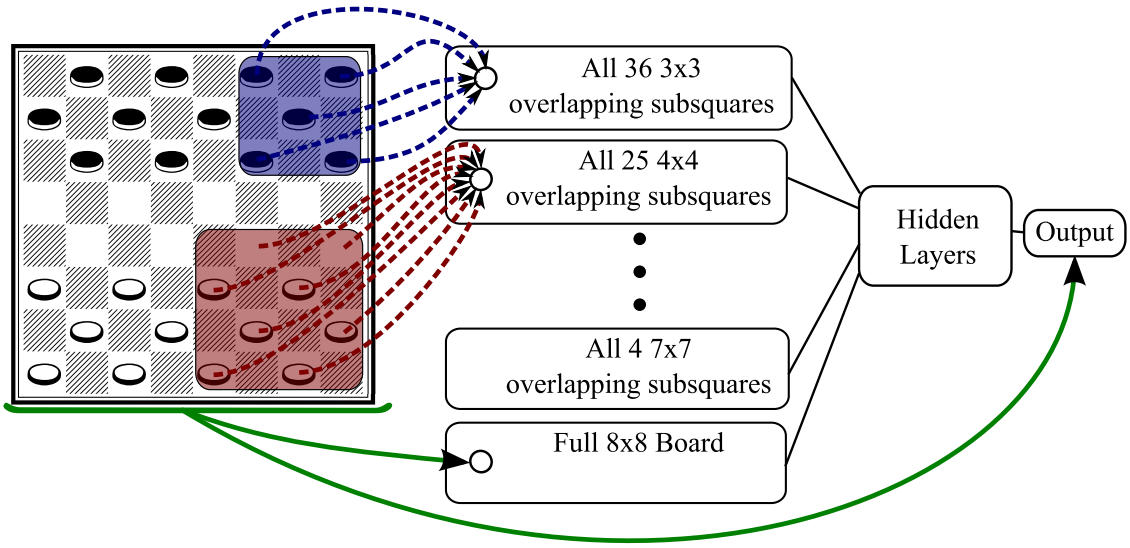


Figure 2.3: **Blondie24 ANN Topology [CF01].** The first hidden layer contains a node for every subsquare of the board of size greater than 2×2. Positions on the board are linked to the corresponding subsquares that contain these positions. This layer then connects to hidden layers that finally connect to the output node. Each valid square on the board connects directly to the output node.

CHAPTER 3

HYPERNEAT

The spatial patterns in Section 2.3 present a challenge: How can such spatial patterns describe connectivity? This chapter explains how CPPN output can be effectively interpreted as a connectivity pattern rather than a spatial pattern in a method called HyperNEAT. HyperNEAT was developed by myself, David D'Ambrosio and Kenneth Stanley at the University of Central Florida [DS07, DS08, GS07a, GS08a, SDG09, GS10a]. The text in this chapter is based on Stanley et al. [SDG09]. This novel representation allows neurons, sensors, and effectors to exploit meaningful geometric relationships. The next section introduces the key insight, which is to assign connectivity a geometric interpretation.

3.1 GEOMETRIC CONNECTIVITY PATTERNS

The main idea behind HyperNEAT is to input into the CPPN the coordinates of the *two points* that define a connection rather than inputting only the position of a single point as in Section 2.3. The output is interpreted as the *weight* of the connection rather than the

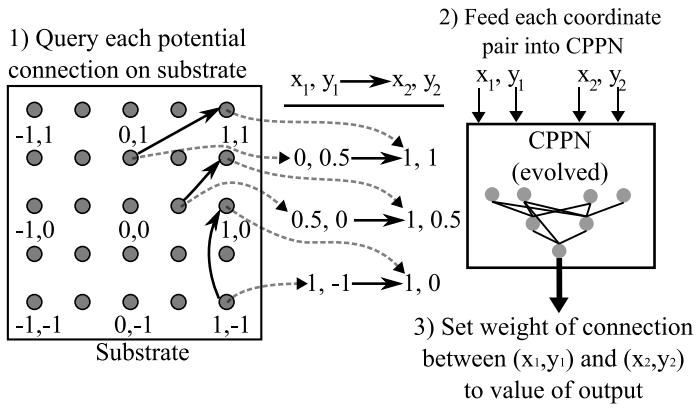


Figure 3.1: **Hypercube-based Geometric Connectivity Pattern Interpretation.** A grid of nodes, called the *substrate*, is assigned coordinates such that the center node is at the origin. (1) Every potential connection in the substrate is queried to determine its presence and weight; the dark directed lines shown in the substrate represent a sample of connections that are queried. (2) For each query, the CPPN takes as input the positions of the two endpoints and (3) outputs the weight of the connection between them. In this way, *connective CPPNs* produce regular patterns of connections in space.

intensity of a point. This way, connections can be defined in terms of the locations that they connect, thereby taking into account the network's geometry.

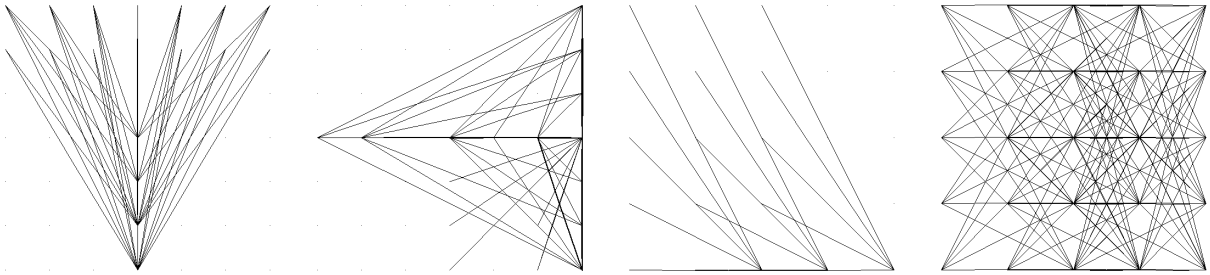
For example, consider a 5×5 grid of nodes. The nodes are assigned coordinates corresponding to their positions within the grid (labeled *substrate* in figure 3.1), where $(0, 0)$ is the center of the grid. Assuming that these nodes and their positions are given *a priori*, a *geometric connectivity pattern* is produced by a CPPN that takes any two coordinates (source and target) as input, and outputs the weight of their connection. The CPPN is queried in this way for every potential connection on the grid. Because the connection weights are thereby a function of the *positions* of their source and target nodes, the distribution of weights on connections throughout the grid will exhibit a pattern that is a function of the geometry of the coordinate system.

A CPPN in effect computes a four-dimensional function $\text{CPPN}(x_1, y_1, x_2, y_2) = w$, where the first node is at (x_1, y_1) and the second node is at (x_2, y_2). This formalism returns a weight for every connection between every node in the grid, including recurrent connections.

By convention, a connection is not expressed if the magnitude of its weight, which may be positive or negative, is below a minimal threshold w_{min}. The magnitude of weights above this threshold are scaled to be between zero and a maximum magnitude in the substrate. That way, the pattern produced by the CPPN can represent any network topology (figure 3.1).

The connectivity pattern produced by a CPPN in this way is called the *substrate* so that it can be verbally distinguished from the CPPN itself, which has its own internal topology. Furthermore, CPPNs that are interpreted to produce connectivity patterns are called *connective CPPNs* while CPPNs that generate spatial patterns are called *spatial CPPNs*. This dissertation focuses on neural substrates produced by connective CPPNs.

Because the CPPN is a function of four dimensions, the two-dimensional connectivity pattern expressed by the CPPN is isomorphic to a spatial pattern embedded in a four-dimensional hypercube. Thus, because CPPNs generate regular spatial patterns (Section 2.3), by extension they can be expected to produce geometric connectivity patterns with corresponding regularities. The next section demonstrates this capability.



(a) Symmetry (b) Imperf. Sym. (c) Repetition (d) Variation

Figure 3.2: **Connectivity Patterns Produced by Connective CPPNs.** These patterns, produced through interactive evolution, exhibit several important connectivity motifs: (a) bilateral symmetry, (b) imperfect symmetry, (c) repetition, and (d) repetition with variation. That these fundamental motifs are compactly represented and easily produced suggests the power of this encoding.

3.2 PRODUCING REGULAR CONNECTIVITY PATTERNS

Simple, easily-discovered substructures in the connective CPPN produce important connective motifs in the substrate. The key difference between connectivity patterns and spatial patterns is that each discrete unit in a connectivity pattern has *two* x values and *two* y values. Thus, for example, symmetry along x can be discovered simply by applying a symmetric function (e.g. Gaussian) to x_1 or x_2 (figure 3.2a).

The human brain is roughly symmetric at a gross resolution, but its symmetry is imperfect. Thus, imperfect symmetry is an important structural motif in ANNs. Connective CPPNs can produce imperfect symmetry by composing *both* symmetric functions of one axis along with an asymmetric coordinate frame such as the axis itself. In this way, the CPPN produces varying degrees of imperfect symmetry (figure 3.2b).

Another important motif in biological brains is repetition, particularly repetition with variation. Just as symmetric functions produce symmetry, periodic functions such as sine produce repetition (figure 3.2c). Patterns with variation are produced by composing a periodic function with a coordinate frame that does not repeat, such as the axis itself (figure 3.2d). Repetitive patterns can also be produced in connectivity as functions of invariant properties between two nodes, such as distance along one axis. Thus, symmetry, imperfect symmetry, repetition, and repetition with variation, key structural motifs in all biological brains [Sta06a], are compactly represented and therefore easily discovered by CPPNs.

3.3 SUBSTRATE CONFIGURATION

CPPNs produce connectivity patterns among nodes on the substrate by querying the CPPN for each pair of points in the substrate to determine the weight of the connection between them. The layout of these nodes can take forms other than the planar grid (figure 3.1) discussed thus far. Different such *substrate configurations* are likely suited to different kinds of problems.

For example, Churchland [Chu86] calls a single two-dimensional sheet of neurons that connects to another two-dimensional sheet a *state-space sandwich*. The sandwich is a restricted three-dimensional structure in which one layer can send connections only in one direction to one other layer. Thus, because of this restriction, it can be expressed by the

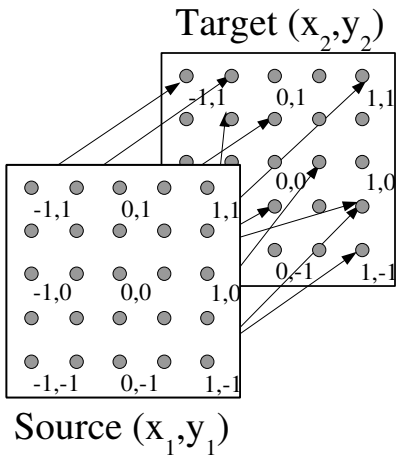


Figure 3.3: **State-Space Sandwich Substrate.** The two-dimensional grid configuration depicted in figure 3.1 is only one of many potential substrate configurations. This figure shows a "state-space sandwich" configuration in which a source sheet of neurons connects directly to a target sheet. Different configurations are likely suited to problems with different geometric properties. The state space sandwich is particularly suited to visual mappings.

single four-dimensional CPPN(x_1, y_1, x_2, y_2), where (x_2, y_2) is interpreted as a location on the *target* sheet rather than as being on the same plane as the source coordinate (x_1,y_1). In this way, CPPNs search for useful patterns within state-space sandwich substrates (figure 3.3), as is done in the experiments in Chapter 6.

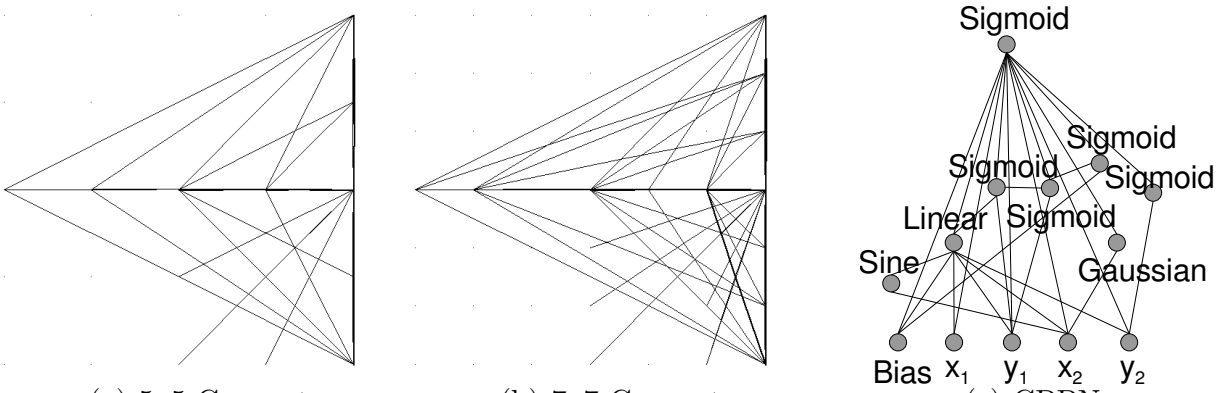
Because connective CPPN substrates are aware of their geometry, they can use this information to their advantage. By arranging neurons in a sensible configuration on the substrate, regularities in the geometry can be exploited by the encoding. Biological neural networks rely on such a capability for many of their functions. For example, neurons in the visual cortex are arranged in the same retinotopic two-dimensional pattern as photoreceptors in the retina [CK04]. That way, they can exploit *locality* by connecting to adjacent neurons with simple, repeating motifs. Connective CPPNs have the same capability.

In fact, when an experimenter arranges the inputs and outputs geometrically according to the task, evolution ceases to be black box optimization because some of the problem structure becomes implicit in the arrangement. As No Free Lunch (NFL) theorems suggest, providing such domain-specific bias can give a performance advanage in some domains. That is, for connective CPPNs with user-placed inputs and outputs, problem-specific bias is provided by the user and therefore any algorithm that can leverage such bias is potentially more effective than those that cannot.

3.4 SUBSTRATE RESOLUTION

As opposed to encoding a specific pattern of connections among a specific set of nodes, connective CPPNs in effect encode a general *connectivity concept*, i.e. the underlying mathematical relationships that produce a particular pattern. The consequence is that *same connective CPPN* can represent an equivalent concept at different resolutions (i.e. different node densities). Figure 3.4 shows a connectivity concept at different resolutions.

For neural substrates, the important implication is that the same ANN can be generated at different resolutions. *Without further evolution*, previously-evolved connective CPPNs can be re-queried to specify the connectivity of the substrate at a new, higher resolution, thereby producing a working solution to the same problem at a higher resolution. This operation, i.e. increasing substrate resolution, introduces a powerful new kind of complexification to ANN



(a) 5x5 Concept (b) 7x7 Concept (c) CPPN

Figure 3.4: **An Equivalent Connectivity Concept at Different Substrate Resolutions.** A connectivity concept is depicted that was evolved through interactive evolution. The CPPN that generates the concept at (a) 5×5 *and* (b) 7×7 is shown in (c). This figure demonstrates that CPPNs represent a mathematical concept rather than a single structure. Thus the same connective CPPN can produce patterns with the same underlying concept at different substrate resolutions (i.e. node densities).

evolution that will be exploited in this dissertation. It is an interesting question whether, at a high level of abstraction, the evolution of brains in biology included several such increases in density on the same connectivity concept. Not only can such an increase improve the immediate resolution of sensation and action, but it can provide additional substrate for increasingly intricate local relationships to be discovered through further evolution.

3.5 COMPUTATIONAL COMPLEXITY

In the most general procedure, a connective CPPN is queried for every potential connection between every node in the substrate. Thus the computational complexity of constructing the final connectivity pattern is a function of the number of nodes. For illustration, consider

a $N \times N$ state-space sandwich. The number of nodes on each plane is N^2. Because every possible pair of nodes is queried, the total number of queries is N^4.

For example, an 11×11 substrate requires 14,641 queries. Such numbers are realistic for modern computers. For example, 250,000 such queries can be computed in 4.64 seconds on a 3.19 Ghz Pentium 4 processor. Thus, generating a population of 100 such networks would take 7.7 minutes. Note that this substrate is an enormous ANNs with up to a quarter-million connections. Connective CPPNs present an opportunity to evolve structures of a complexity and functional sophistication genuinely commensurate with available processing power.

Tactical and strategic sequential decision domains can benefit from HyperNEAT's ability to generate geometric connectivity patterns. Often, the same situation might exist at several locations in the domain geometry simultaneously. A policy based on domain geometry thus can understand and react to similar situations that occur in different areas or configurations. The following chapter explores this potential.

CHAPTER 4

CHECKERS BOARD EVALUATION EXPERIMENT

The experiment in this chapter, which I published with Kenneth Stanley at the Twenty-Third Conference on Artificial Intelligence (AAAI-2008) [GS08a] and extended in Neural Computation journal [GS10a], aims to determine whether encoding geometry helps machine learning to generalize. The idea is to learn to defeat a single fixed training opponent and then test for generalization against variations of this opponent.

4.1 APPROACH: LEARNING REGULARITIES IN CHECKERS

The game of checkers is chosen for the experiments in this chapter to establish HyperNEAT's potential in tactical and strategic decision domains because checkers is intuitively geometric. While approaches like Blondie24 [Fog02] engineer geometry into the ANN topology in the hope that such engineering may be useful, the idea in HyperNEAT is to *learn* from geometry by generating the policy network as a direct function of task geometry. This section explains how that is done in the game of checkers.

To apply HyperNEAT to checkers, the substrate input layer is arranged in two dimensions to match the geometry of the checkers board (figure 4.1). Notice that the substrate in figure 4.1 includes a hidden layer. Thus, it is analogous to two sandwich substrates (e.g. figure 3.3) stacked on top of each other. In particular, the two-dimensional input layer connects to an analogous two-dimensional hidden layer so that the hidden layer can learn to process localized geometric features. The hidden layer then connects to a single output node, whose role is to evaluate board positions. The CPPN distinguishes the set of connections between the inputs and the hidden layer from those between the hidden layer and the output node by querying the weights of each set of connections from a *separate* output on the CPPN (note the two outputs in the CPPN depiction in figure 4.1). That way, the x and y positions of each node are sufficient to identify the queried connection and the outputs differentiate one connection layer from the next. Because the CPPN can effectively compute connection weights as a function of the *difference* in positions of two nodes, it can easily map a repeating concept across the whole board.

In this way, the substrate is a board evaluation function. The function inputs a board position and outputs its value for black. To evaluate the board when it is white's turn to move, the color of the pieces can be reversed and then the sign of the result inverted. To decide which move to make, a minimax search algorithm runs to a fixed ply depth of four. Alpha-beta pruning [KM75] and iterative deepening [RNC95] techniques increase performance without changing the output. The output of the substrate is the heuristic score for the minimax algorithm.

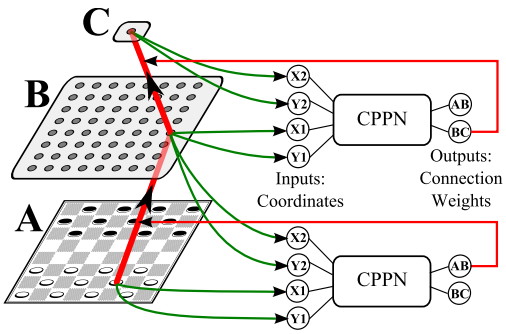


Figure 4.1: **Checkers Substrate.** The substrate (at left) contains a two-dimensional input layer (A) that corresponds to the geometry of a game board, an analogous two-dimensional hidden layer (B), and a single-node output layer (C) that returns a board evaluation. The two CPPNs (at right) are depictions of the *same* CPPN being queried to determine the weights of two different substrate connections. The bottom CPPN depiction receives as input the x and y coordinates of a node in A and a node in B and returns the weight of this connection from its AB output node. Similarly, the top depiction of the same CPPN is being queried for the weight of a connection between B and C and therefore returns this weight from its BC output. In this way, a four-input CPPN can specify the connection weights of a two-layer network structure as a function of the positions, and hence the geometry, of each node.

This approach allows HyperNEAT to discover geometric regularities on the board by expressing connection weights as a function of geometry. It is therefore unnecessary to manually engineer the network topology, or divide the input space into subsections in an attempt to inject a priori theories about the key regularities in the game into the representation. Because HyperNEAT discovers geometric relationships on its own, an *identical substrate* could be applied to other board games even without knowledge of the game rules, contributing to the generality of the approach.

4.2 EXPERIMENTAL DESIGN

The experiment is designed to investigate the role of neural geometry in solving a problem that is clearly geometric. The idea is to learn to defeat a single fixed training opponent and then test for generalization against variations of this opponent. Thus rather than producing the best possible checkers player, the aim is to analyze in detail the implications of a geometric representation, not only for learning, but especially for generalization beyond what was trained.

Board games are an effective platform to discern the importance of geometry because they depend heavily on geometric relationships that often repeat across the board. Therefore, to begin the investigation, this chapter compares four evolutionary approaches that take geometry into account to varying degrees in the domain of checkers. Each approach is trained against the same hand-engineered deterministic opponent [Fie02]. The opponent is a linear combination of several heuristics, including material possession, positional bias, whether pieces on the back row have been moved (which would lower the score), whether a double corner is intact, and who controls the center and the edge of the board. Thus the deterministic opponent is nontrivial, i.e. not just a simple piece counter. During evolution, each candidate plays a single game as black against the opponent to determine its fitness. Both the evolved player and the opponent evaluate boards that are four ply ahead. Fitness is computed as a function of both the final game state and intermediate board states. At

each turn t, fitness f_t is awarded based on the current board state according to the equation:

$$f_t = 100 + 2m_s + 3k_s + 2(12 - m_o) + 3(12 - k_o), \tag{4.1}$$

where m_s and m_o are the number of regular pieces possessed by the learner and the opponent, respectively, and k_s and k_o are the number of kings. The coefficients 2 and 3 represent the values of pieces and kings respectively, denoting that kings are roughly 1.5 times as valuable as regular pieces. Because there are at most 12 pieces of any given type, the number 12 ensures positive values for its respective terms. This function rewards incremental progress and provides a smoother learning gradient than simply awarding fitness based on the final score. The value of 100 per turn rewards individuals more who play games that last a longer number of turns. Thus, evolved players that lose quickly will receive less fitness. If the evolved player wins, fitness is awarded over 100 turns, even if the game ends earlier. That way, winning early is not penalized. If the candidate wins against the training opponent, an additional 30,000 is added to the total fitness. It is important to note that this fitness function is unique and not based on Blondie24 [Fog02], whose results are therefore not directly comparable.

The learned strategies are then tested against a non-deterministic variant of the same opponent. This variant has a 10% chance of choosing the second-highest scoring move instead of the optimal move found in minimax search. This approach is similar to work done by Fogel [Fog93], who also implemented a percent chance of picking a random move to diversify a

deterministic opponent. Methods that evolve more general solutions should produce policies that win more such games.

The four compared approaches are chosen carefully to isolate the issue of geometric processing. Therefore, they are all variants of the same NeuroEvolution of Augmenting Topologies (NEAT) approach (Section 2.4.1). This shared basis means that differences in performance are attributable to the way each approach processes its inputs. For all four approaches, input values of 0.5 and -0.5 encode black and white pieces, respectively. Kings are represented by a magnitude of 0.75, which is similar to the approach in Chellapilla and Fogel [CF01], who showed that multiplying the standard piece input magnitude by 1.3 produces a good magnitude for kings in their approach. A single output expresses the value of the current board state for black.

Regular NEAT inputs a vector of length 32 in which each parameter represents a square on the board that can potentially hold a piece. NEAT evolves the topology and weights between the input and output nodes.

NEAT-EI is an attempt to enhance NEAT's ability to take into account geometric regularities across the board by supplying additional engineered inputs (EI). It has the same inputs as NEAT; however, the starting network topology is engineered as in Blondie24 to have additional inputs that focus on geometric regions of differing sizes (CF01; figure 2.3). The result of training NEAT-EI in this chapter cannot be compared directly to Blondie24 because Blondie24 is the result of *coevolution* while the policies in this chapter are evolved against a fixed opponent. Rather than evolving the best possible player, the goal in this

chapter is to fairly compare the generalization of different representations by teaching each to defeat an identical opponent, thereby isolating the issue of generalization.

HyperNEAT inputs are arranged in a two-dimensional 8×8 grid that forms the first layer of a three-layer substrate (figure 4.1). Pieces are placed on this grid in literally the same position that they exist in the domain. The second layer is an 8×8 hidden layer, and the third layer is a single node that returns the board evaluation.

For HyperNEAT, NEAT evolves the CPPN that computes the connection weights of the substrate.

If it is indeed possible to exploit geometry to improve play, the better an approach can represent geometric relationships (either through learning or a priori engineering), the better that method should learn and generalize.

FT-NEAT (fixed-topology NEAT) inputs are arranged in the same configuration as in the substrate in HyperNEAT (figure 4.1). FT-NEAT evolves the weights of this ANN but not the topology. Thus FT-NEAT must evolve the connection weights of over 4,000 *directly-encoded* connections, helping to confirm that it is not just the particular topology of the substrate in figure 4.1, but more importantly the indirect encoding in HyperNEAT, that provides an advantage.

After the experimental comparison among the four methods, an extensive analysis of substrate visualizations from more and less general HyperNEAT-evolved players investigates

how geometry influences generalization, and the way evolved maps are organized. The parameters for this experiment are provided in Appendix A.

4.3 RESULTS

Performance in this section is measured in two ways. First, the fitness of each approach is tracked during training over generations, which gives a sense of relative *training* performance. Second, after training is complete, the best solutions from each run play 100 games against the randomized opponent, yielding generalization. The main question is whether HyperNEAT's ability to learn from geometry benefits its performance and generalization.

4.3.1 TRAINING PERFORMANCE

Figure 4.2 shows the average generation champion fitness over evolution, averaged over 20 runs. While *none* of the runs of regular NEAT nor FT-NEAT were able to defeat the opponent within 200 generations, both HyperNEAT and NEAT-EI learned to defeat it in all runs. On average, it took NEAT-EI 57.9 generations to find a winning solution. HyperNEAT succeeds much more quickly, finding a winner in 8.2 generations on average. These differences are statistically significant according to Student's t-test ($p < 0.05$). This disparity highlights the critical importance of learning from geometry. While defeating the heuristic appears

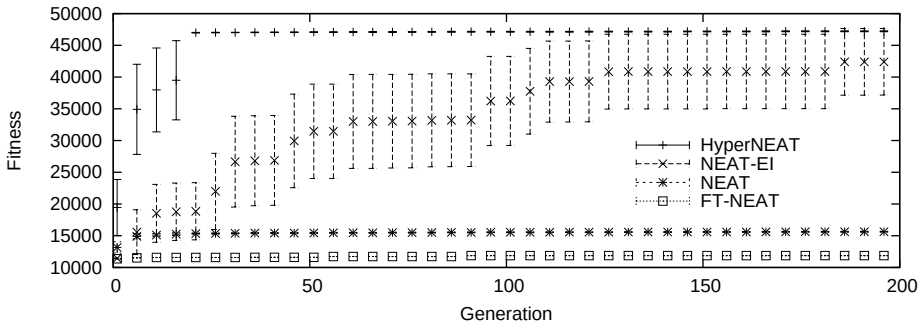


Figure 4.2: **Fitness During Training.** The fitness of the generation champions of each approach is shown, averaged over 20 runs. HyperNEAT generation champions perform significantly better than NEAT-EI between generations 1 and 123 ($p < .05$ using Student's t-test). Error bars show the 95% confidence interval. HyperNEAT learns faster than NEAT-EI because its CPPN solutions require fewer dimensions to represent.

challenging with direct representations, it becomes easy if the solution is learned as a function of the board geometry.

4.3.2 GENERALIZATION

Every generation champion that defeats the deterministic opponent plays 100 games against the randomized opponent. Because regular NEAT and FT-NEAT could never defeat this opponent, they are not included in this test. To make the comparison fair, only the *most general* solutions of each run are compared, which means the generation champion with the highest score computed by $W + \frac{T}{2}$, where W and T are the number of wins and ties against the randomized opponent, respectively. The equation $W + \frac{T}{2}$ is used to convert a wins, losses, and ties metric to a single scalar score. That way, the generalization results focus on the *best possible* generalization for both methods when they learn to defeat an

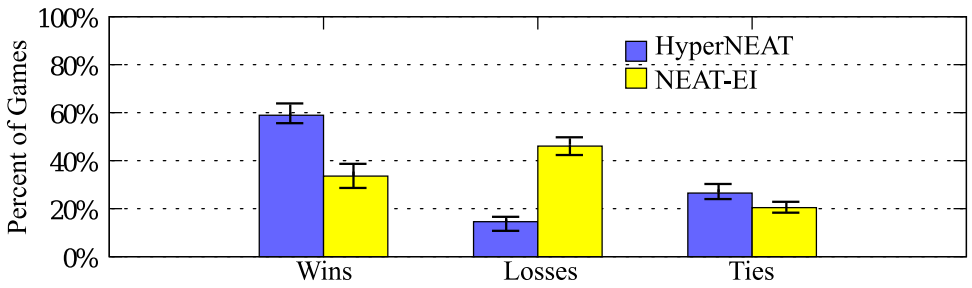


Figure 4.3: **Generalization Results.** Average wins, losses, and ties in 100 games against the randomized opponent are shown for HyperNEAT and NEAT-EI, averaged over 20 runs of each. Only the most general solutions of each run are included in the test. HyperNEAT solutions win significantly more games ($p < 0.05$) and lose significantly fewer games ($p < 0.05$) than NEAT-EI. Error bars show a 95% confidence interval. The difference in ties between the two methods is not significant ($p \approx 0.06$).

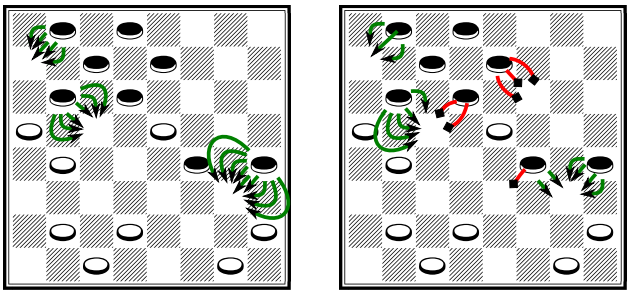
identical opponent. The best possible generalization represents what would result from an ideal validation of the trained opponents. While in the real world such idealized validation may not always be possible, assuming reasonable effort on the part of the experimenter, it is a yardstick for how well a system can be expected to perform in a reinforcement learning task. A similar approach to measuring generalization in such a task is taken by Gruau et al. [GWP96]. Figure 4.3 shows the results of these solutions against the randomized opponent. HyperNEAT wins significantly more and loses significantly less than NEAT-EI. The geometric encoding allows HyperNEAT to generalize across the board.

4.3.3 TYPICAL SOLUTIONS

HyperNEAT's advantage is most evident in the middle-game and later. As the game-tree branches, deviation from the training opponent increases. Because HyperNEAT performs

better in such novel situations, it is more general. For example, figure 4.4 contrasts moves chosen by NEAT-EI solutions with those from HyperNEAT from the *same* unfamiliar position. NEAT-EI players unnecessarily sacrifice pieces, while HyperNEAT players rarely do from this position. Given that the evaluations during training consist of a single game against a deterministic opponent, the ability of a solution evolved during training to perform well in generalization tests against a non-deterministic opponent is significant. These typical solutions demonstrate the idea that, because HyperNEAT evolves a pattern of weights across the geometry of the substrate, HyperNEAT is able to evolve a player that can both defeat the deterministic heuristic and simultaneously perform well in generalization tests, without any need for generalization pressure in the fitness function. Conversely, NEAT-EI struggles to generalize, suggesting that NEAT-EI learned a specific subset of board states instead of a general checkers strategy. In the case of NEAT-EI, generalization would likely benefit from playing additional games in a single evaluation.

The most general solution in all runs of NEAT-EI has 126 nodes and 1,106 connections. In contrast, the most general solution of HyperNEAT is a CPPN with only 23 nodes and 84 connections, which generates an ANN with 129 nodes and 3,979 connections. Figure 4.5 illustrates this dramatic compression afforded by indirect encoding in a typical HyperNEAT solution. In this way, HyperNEAT is able to explore a significantly smaller search space (i.e. CPPNs) while still creating complex structures (i.e. substrates).



(a) HyperNEAT Moves (b) NEAT-EI Moves

Figure 4.4: **Requested moves from the same board position by HyperNEAT and NEAT-EI.** This figure depicts a position several moves into a game. Twenty moves requested by the champions of all NEAT-EI runs are contrasted with twenty from HyperNEAT runs. All of the HyperNEAT runs suggest neutral or positive moves. Six of the NEAT-EI runs make moves that lead to immediate, uncompensated loss. These moves are denoted with a darker line and a square endpoint.

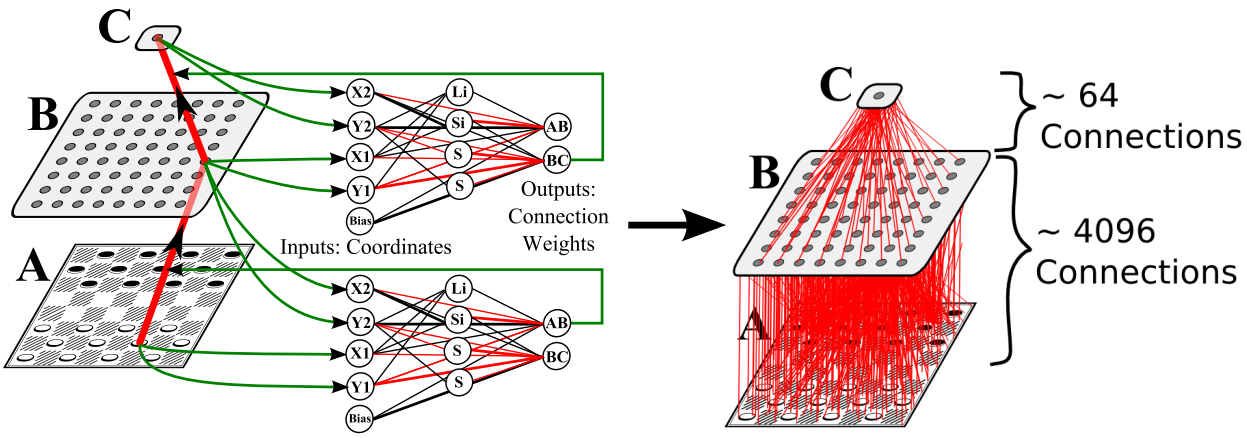


Figure 4.5: **Compression in CPPN Encoding.** The CPPN at left, which is an actual solution against the heuristic, contains only 18 connections, yet it encodes the connection weights of a substrate with over 4,000 connections. In this way, HyperNEAT searches a significantly lower-dimensional space than a direct encoding. In the figure above, the letters A, B, and C represent the input, hidden, and output layer, respectively. The output labeled "AB" determines the connection weight for a link originating in the input layer and terminating at the hidden layer (following figure 4.1).

4.3.4 SUBSTRATE VISUALIZATIONS

While the results so far establish that learning from geometry provides an advantage in both performance and generalization, an important question is how exactly this advantage is realized. This section aims to investigate this question by examining the internal connectivity and activation patterns of HyperNEAT-trained networks. It is important to note that this study of the topographic layout of nodes and connectivity within an evolved ANN is only possible because, unlike other neuroevolution algorithms [ASP93, KR91, SM02, IJC89, SP91, FDM08, Yao99, GWP96, Kit90, Fog93, BG92, Sim94, LP00, HP02, GM97, MM96, OS97, ZM93, BW93, PP98, Bon02], the neurons within a HyperNEAT substrate are situated at geometric coordinates. This geometry is what affords the opportunity to observe patterns in their actual situated geometric context, giving insight into why such a context is important to learning in general and what kinds of opportunities it creates.

The particular focus of the analysis in this section is on the question of what kind of connectivity patterns lead to generalization and what kind do not. To investigate this difference, a group of four highly general HyperNEAT solutions and four HyperNEAT solutions that generalize less effectively (summarized in table 4.1) are visualized in two different ways:

First, the connectivity patterns of the solutions are visualized through images of influence maps and receptive fields. These images are arranged vertically within a single panel in one column (figure 4.6). The bottom image of each panel is a set of five *influence maps* that shows how individual inputs from the checkers board influence the entire hidden layer.

Solution	Wins	Losses	Ties
1	53	22	25
2	62	17	21
3	61	17	22
4	54	28	18

(a) General solutions against non-deterministic heuristic

Solution	Wins	Losses	Ties
1	35	24	41
2	39	8	53
3	38	33	29
4	35	17	48

(b) Less general solutions against non-deterministic heuristic

Table 4.1: **Selected general and less general HyperNEAT solutions.** The two tables show the wins, losses, and ties of selected (a) general and (b) less general champions against the non-deterministic heuristic that are visualized later in this section. Note that the champions selected are not necessarily the champions of the run. Because the training phase involves only a single game against a deterministic heuristic, there is no explicit reward for generality in the fitness function. Even so, some of the runs produce solutions that generalize better than others. Note that because HyperNEAT generalizes well on average, the poorest generalizers from HyperNEAT still out-generalize average NEAT-EI champions significantly. Nevertheless, the difference between these less general champions and those that are even more general still helps to elucidate the factors underlying effective generalization.

The intensity at each position within each such map represents the magnitude of a single connection weight, and white triangles in the top-left corner of a position represent negative connection weights (i.e. darker color denotes less influence). Thus a full influence map shows all the weights projecting from a single input to the entire hidden layer. The five influence maps form a cross shape, symbolizing that they represent images coming from five locations on the checkers board, as shown in figure 4.6. Above the five influence maps are a similar five *receptive field* visualizations. These images are designed to show how each *hidden node* sees all of the inputs that can connect to it. Like the influence maps, the five receptive field visualizations are also shown in a cross, in this case to represent where in the *hidden layer* the receiving node is located. The single image at the top is the receptive field of the single

output node, which shows the connection weights from the hidden layer to the output, which is how the final computation of the board value is completed.

Second, visualizations of hidden layer *activation patterns* for several board positions illustrate how boards are evaluated in the game-tree (figure 4.7). Each figure with this type of visualization displays a row of activation patterns obtained by the champion of a particular run from table 4.1. The activation pattern is illustrated by a column depicting the input board configuration (bottom), the hidden layer node activation levels (middle), and the output activation (top), which is the value assigned to that board position for black.

These activation patterns are also organized topographically such that the activation level of each hidden neuron is depicted at that neuron's actual position in the substrate. Thus it is possible to see how the activation levels relate to the network's geometry. Each board position is an actual position encountered during alpha-beta search, so the overall visualization makes it possible to see how the network represents the difference between relatively good or bad situations. The board positions and activation patterns are ordered from left to right by increasing output value so that it is easy to see how increasingly good positions are represented by the network internally. Note that the sequence from left to right depicts board positions that were encountered during alpha-beta search rather than a sequence of moves during actual gameplay. The aim is to elucidate how judgments on board quality are represented.

The contrast between connectivity patterns from general (figure 4.8) and less general (figure 4.9) players is surprisingly revealing. In fact, the distinguishing characteristic of

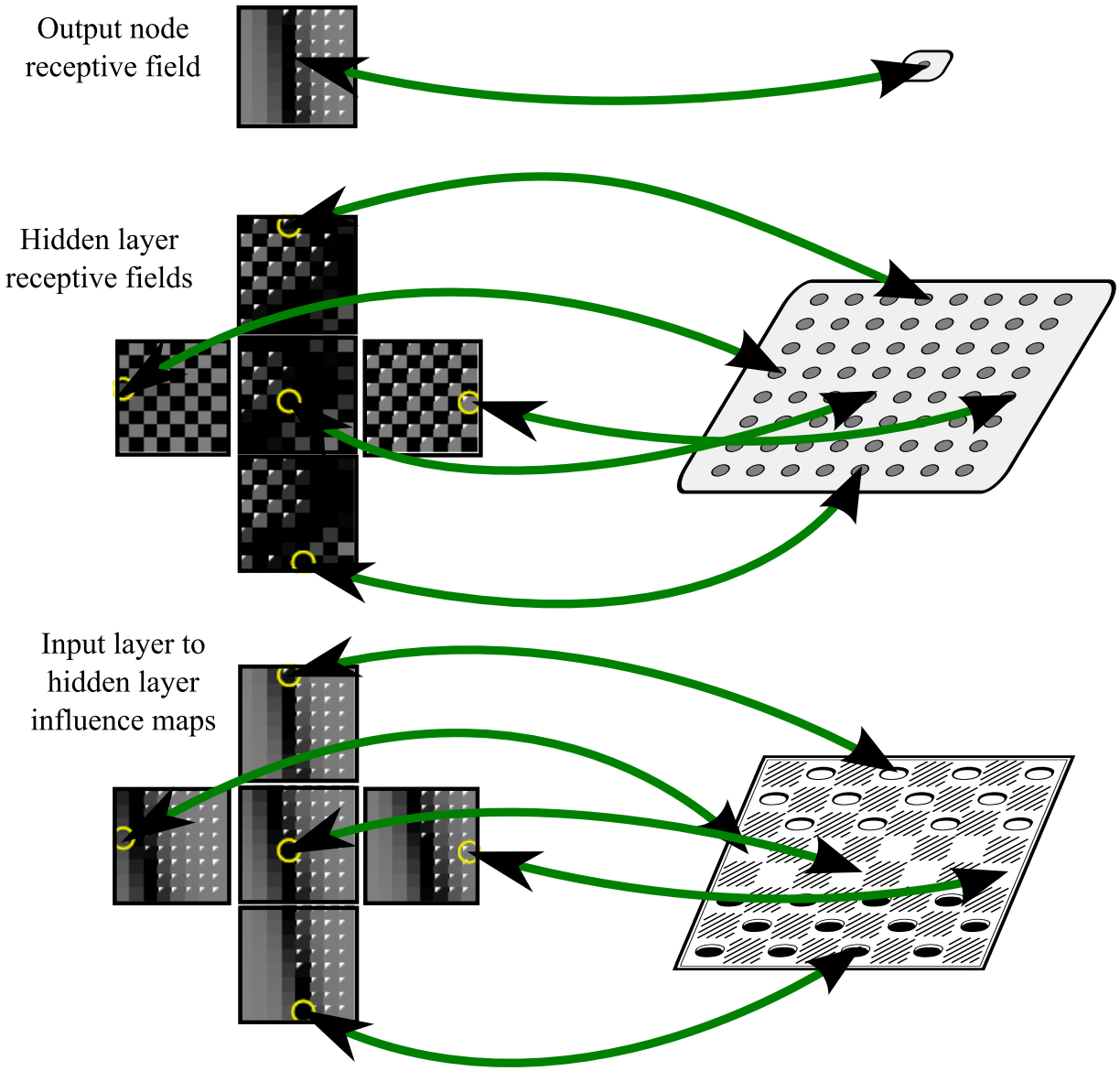


Figure 4.6: **Visualizing connection weights.** In this section (figures 4.8 and 4.9), connection weights within substrates are depicted as shown in this figure. For the *influence maps* (bottom), the lines show from which square on the checkers board each influence map originates. Similarly, *receptive fields* (middle and top) are shown for the hidden nodes with which each such field is connected. An influence map originates *from a single input*, and a receptive field terminates *at a single hidden node*. The cross patterns are designed to make it easy to see how the pattern of each influence map or receptive field varies with their originating position.

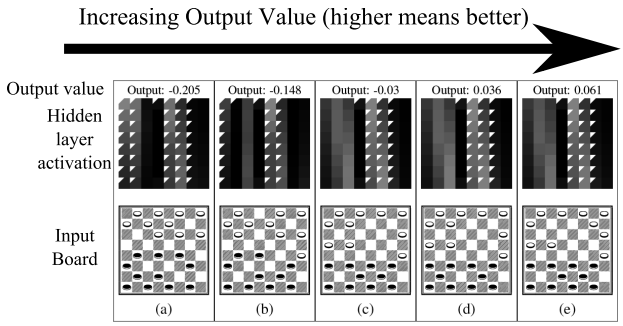


Figure 4.7: **Visualizing hidden layer activation patterns for different board positions.** To understand how different quality board positions influence the hidden layer of a particular general or less general substrate, board positions and hidden layer activations in this section (figures 4.10–4.17) are visualized as shown in this figure. Each such board position was actually encountered by alpha-beta search with the associated substrate during gameplay. To elucidate how the hidden layer distinguishes worse from better positions, panels (a) through (e) are always ordered from the lowest evaluation score returned by the substrate output to the highest. That way, it is possible to understand the scenarios preferred by the learned evaluation function.

general play is visually apparent by simply observing its geometry: The connectivity patterns of networks that generalize most effectively exhibit *smooth* boundaries while the boundaries of those that generalize poorly are *jagged*. This difference is particularly prominent in the influence maps (at the bottom of figures 4.8 and 4.9), suggesting that influence maps from inputs reveal an important facet of geometry. These characteristics are consistent across all general and less general solutions in figures 4.8 and 4.9. Thus there is a strong correlation between generality and geometric smoothness.

The role of smoothness in generalization yields the important insight that the ability to *represent* smooth regularity is a critical prerequisite to consistent generalization. In contrast, jaggedness suggests *memorization* of the specific situations encountered while playing the particular opponent heuristic (recall that both general and less general solutions defeated

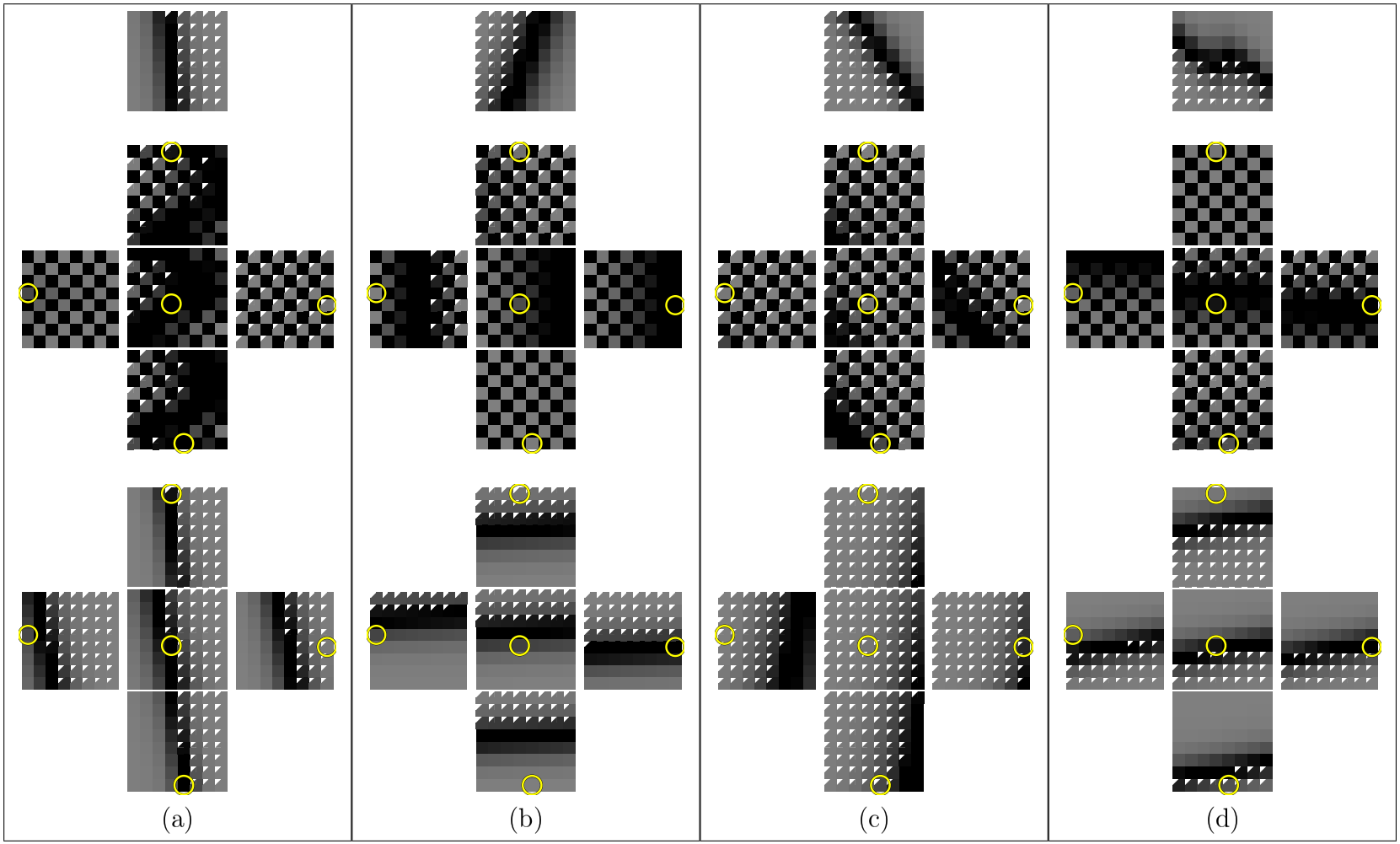


Figure 4.8: **Connectivity patterns of general solutions.** Influence maps and receptive fields (as explained in figure 4.6) are shown for four general solutions. An important feature shared by all general solutions is that their connectivity patterns are smooth and continuous. They also vary in a regular fashion with their originating node's location (bottom) or hidden node location (middle).

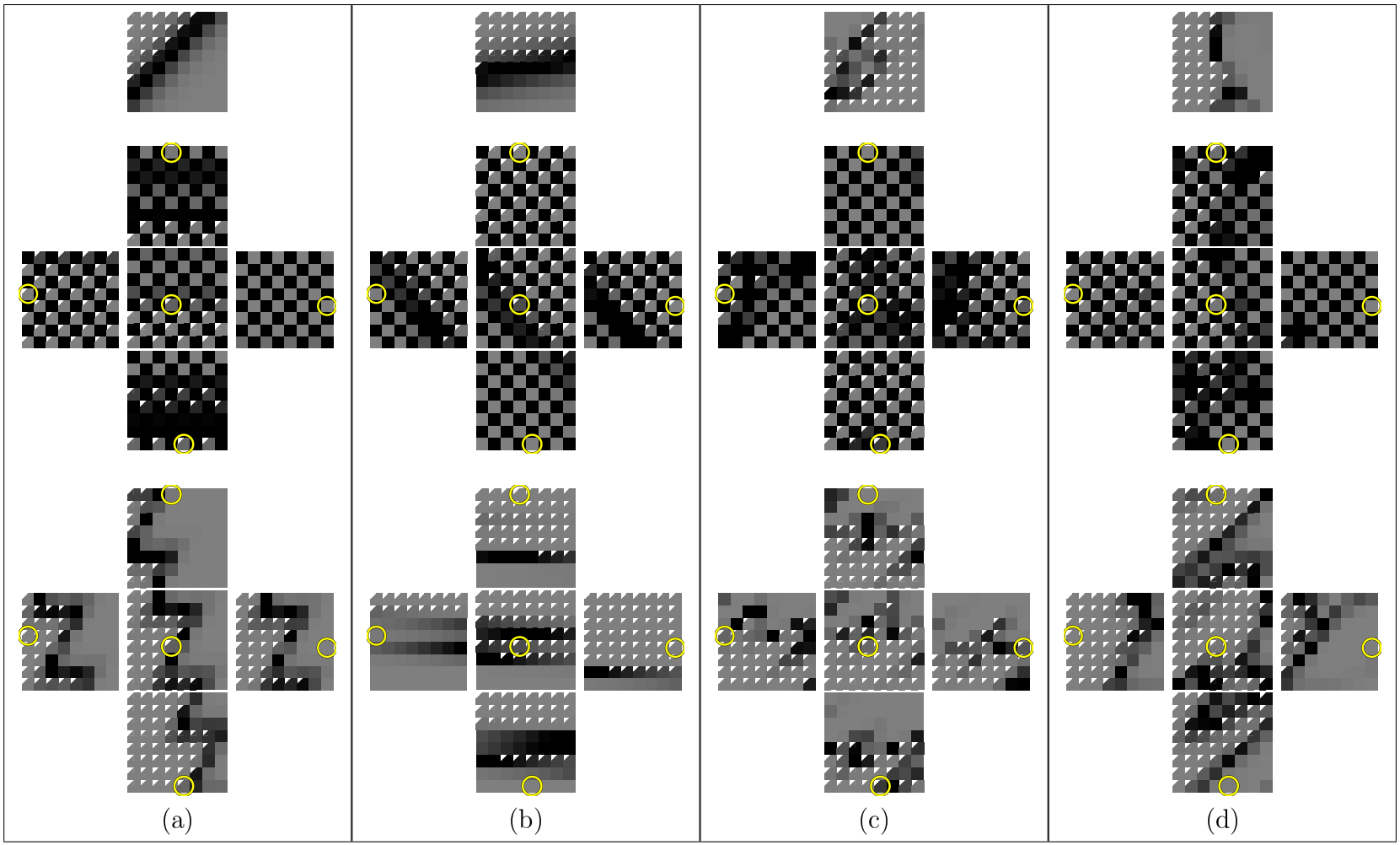


Figure 4.9: **Connectivity patterns of less general solutions.** The influence maps and receptive fields shown in this figure are for solutions that are less general than those in figure 4.8. Interestingly, the connectivity patterns of less general solutions are markedly jagged and discontinuous. This property is indicative of overspecialization to the training heuristic

the deterministic heuristic during training). The irregularity of the jagged solutions, which nevertheless beat the training heuristic, is an artifact of the peculiarities of the heuristic itself and not always useful when playing even slightly altered strategies.

Interestingly, smooth regularity is *only* natural to represent in a geometric context. After all, the connectivity that emanates from each input neuron in figure 4.8 varies in a regular geometric fashion as the source neuron shifts position across the board (observe the pattern at different locations within each cross). Only an indirect encoding that describes connectivity as a function of geometry is likely to consistently yield such regularities. A direct encoding, on the other hand, cannot *describe* how a pattern varies smoothly over space. Therefore, in a direct encoding, each individual connection is learned separately, most likely yielding jagged patterns. (Note that because direct encodings do not have a geometry, they cannot be visualized in this way; however, it is exactly this fact that prevents them from expressing smooth patterns that vary across geometry.) In fact, if the patterns yielded by direct encodings could be situated geometrically, they would likely be significantly more irregular than even those in figure 4.9, which still are at least the product of indirect encoding.

In effect, the patterns in figure 4.8 are *evolved topographic maps* for the game of checkers. A special (and unique) kind of receptive field is evolved in each case that moves in a predictable regular fashion across the hidden layer in accordance with the position of the source neuron from the input layer. In the less general solutions (figure 4.9), these maps are less regular and more distorted, hurting generalization.

Another important observation about the general patterns in figure 4.8 is that while they are all regular and smooth, they are also all different from each other. That is, the receptive field structure in figure 4.8a is unlike figure 4.8b,c,d, figure 4.8b is unlike 4.8a,c,d, etc. Therefore, interestingly, the implication is that as long as smoothness and regularity are achieved in the influence maps, there are multiple ways to solve the same problem effectively, perhaps explaining why HyperNEAT beats the heuristic so quickly while still generalizing often. It is the bias towards smooth topographic maps that portends the ability to generalize.

As explained in figure 4.7, figures 4.10 through 4.17 show how these connectivity patterns, both general and less general, integrate to evaluate actual board positions encountered in the game-tree. It is important to note that the precise value of the output node activation (shown at the top of each panel) is not important; because evaluations are performed within an alpha-beta search, only the relative output impacts decision-making. For example, a negative output value does not necessarily indicate a poor evaluation and vice versa. It is also important to note that differences in board positions are often reflected in subtle changes in neural activation among hidden nodes. Thus it is occasionally difficult to perceive these changes visually. Nevertheless, they are often perceptible by comparing activation patterns closely, and their subtlety signifies the precision with which the substrate disambiguates similar board positions.

Through figures 4.10–4.17, it is once again clear that there are many ways to solve checkers against the heuristic. However, it is also apparent that the overall activation patterns on the hidden layer exhibit definite *shapes* that are tied inextricably to the geometry of the board

itself. That is, areas of high activation are normally all adjacent and separated from areas of low activation, even if these areas appear in different locations for different solutions. Thus the overall activation pattern, which is realized through the individual neural connectivity patterns in figure 4.8, is also fundamentally geometric, combining the joint assessments of each individual receptive field.

Less general solutions (figures 4.14–4.17) yield activation patterns that are mainly jagged and discontinuous. These shapes do not resemble general solutions (figures 4.10–4.13), although they still exhibit patterns that are geometric. The jaggedness in the patterns suggests that the solutions are able to defeat the deterministic heuristic by memorizing certain states, and not by encoding a holistic pattern that describes the dynamics of checkers.

The general solutions (figures 4.10–4.13) typically favor a holistic strategy. For example, in figures 4.10 and 4.11, keeping pieces in a tightly bound group at the back of the board is rewarded, although the lateral focus of density (i.e. left versus right) differs. In contrast, the substrates in figures 4.11 and 4.12 favor solutions that are more aggressive and attempt to control of the center of the board. Nevertheless, the principle that unifies all these approaches is their generality; they are sensitive to relative concentrations of groupings of pieces.

Less general solutions (figures 4.14–4.17), while often reasonable, exhibit idiosyncratic holes in their approach that are reflected in their more piecemeal activation patterns. As described in figures 4.14–4.17, such idiosyncrasies often yield specific evaluations that are fundamentally flawed. For example, position 4.15d is rated relatively highly, yet leaves black open to a double-jump on the next turn, after black takes white's piece. Position 4.17e

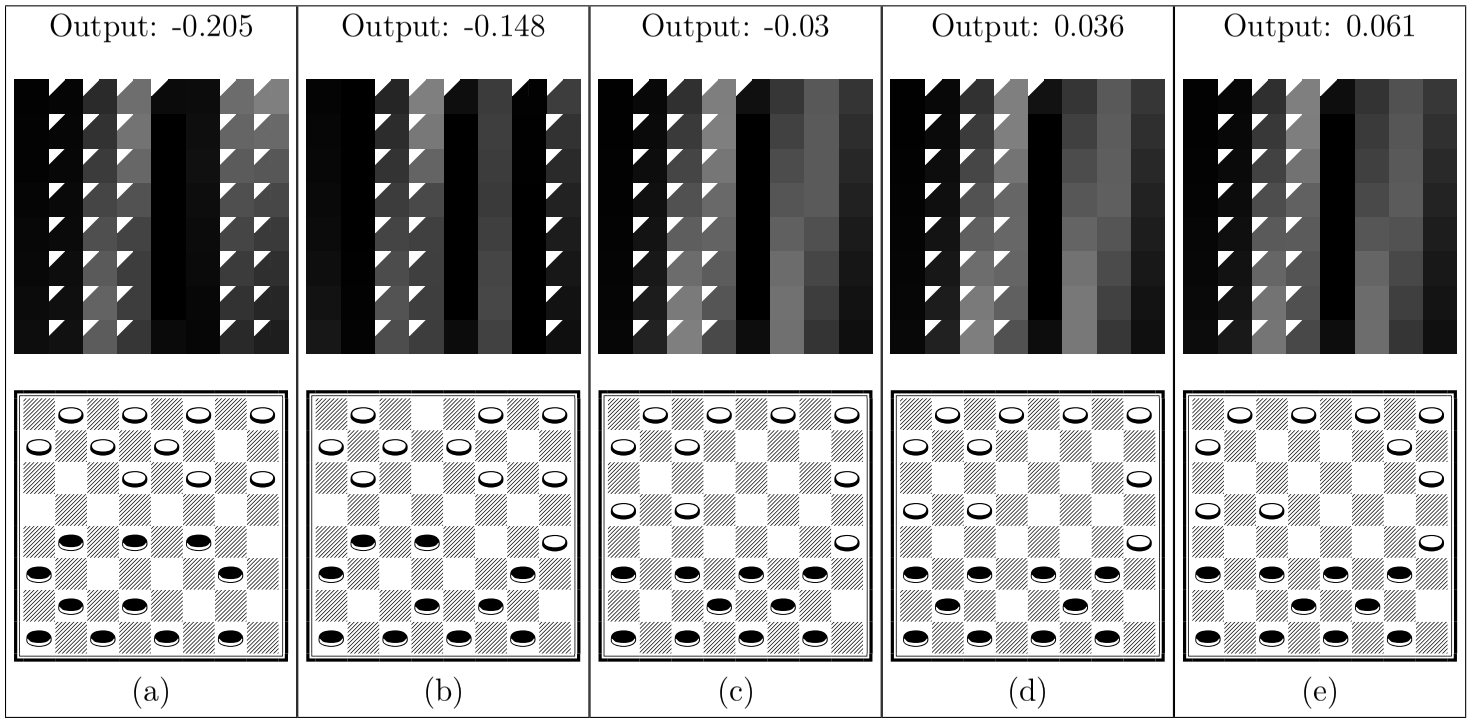


Figure 4.10: **Board positions and associated hidden layer activation patterns encountered by alpha-beta search for the general solution shown in figure 4.8a.** This substrate prefers black density in the lower-right sector of the board. As a result, black aims to bunch into a group. This structure prevents any single piece from being taken.

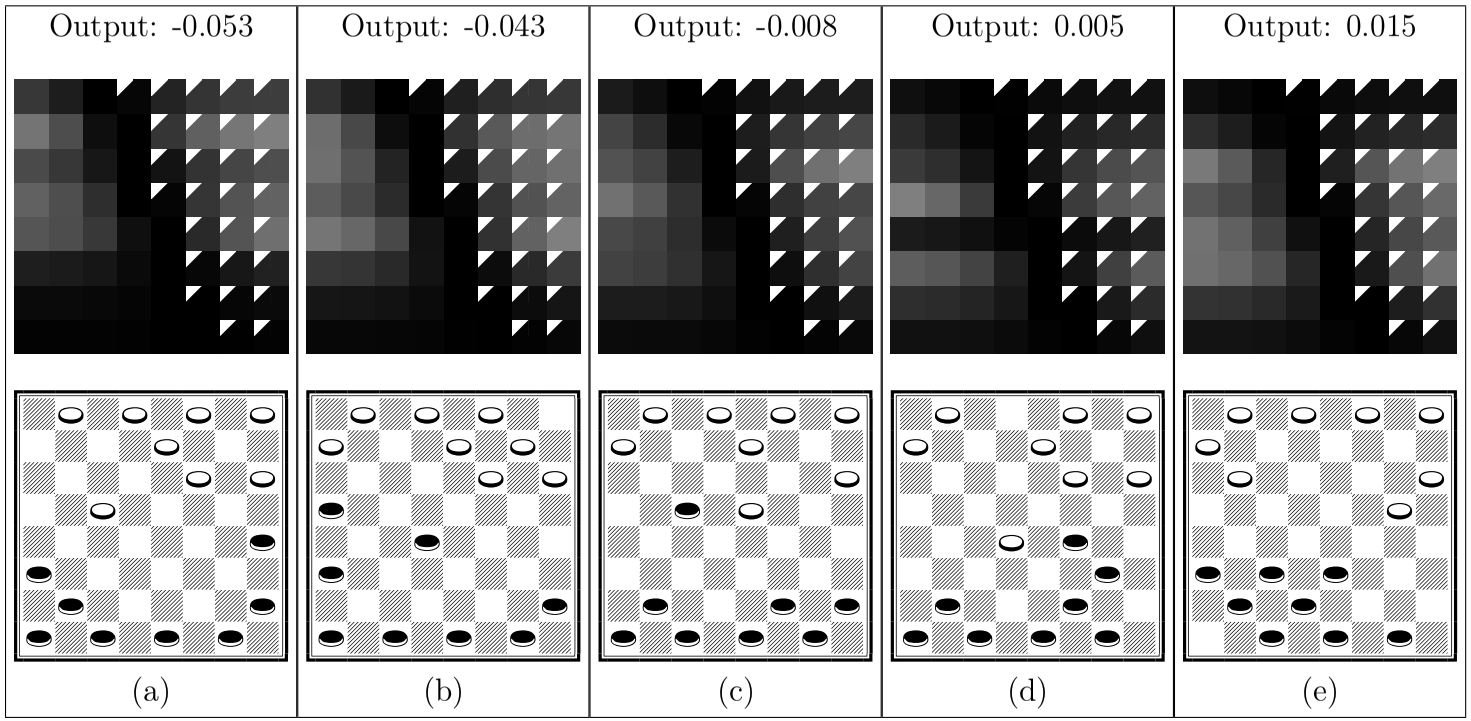


Figure 4.11: **Board positions and associated hidden layer activation patterns encountered by alpha-beta search for the general solution shown in figure 4.8b.** Like the solution in figure 4.10, this substrate also prefers a defensive stance with density in the back rows. However, unlike in figure 4.10, this strategy prefers density on the left.

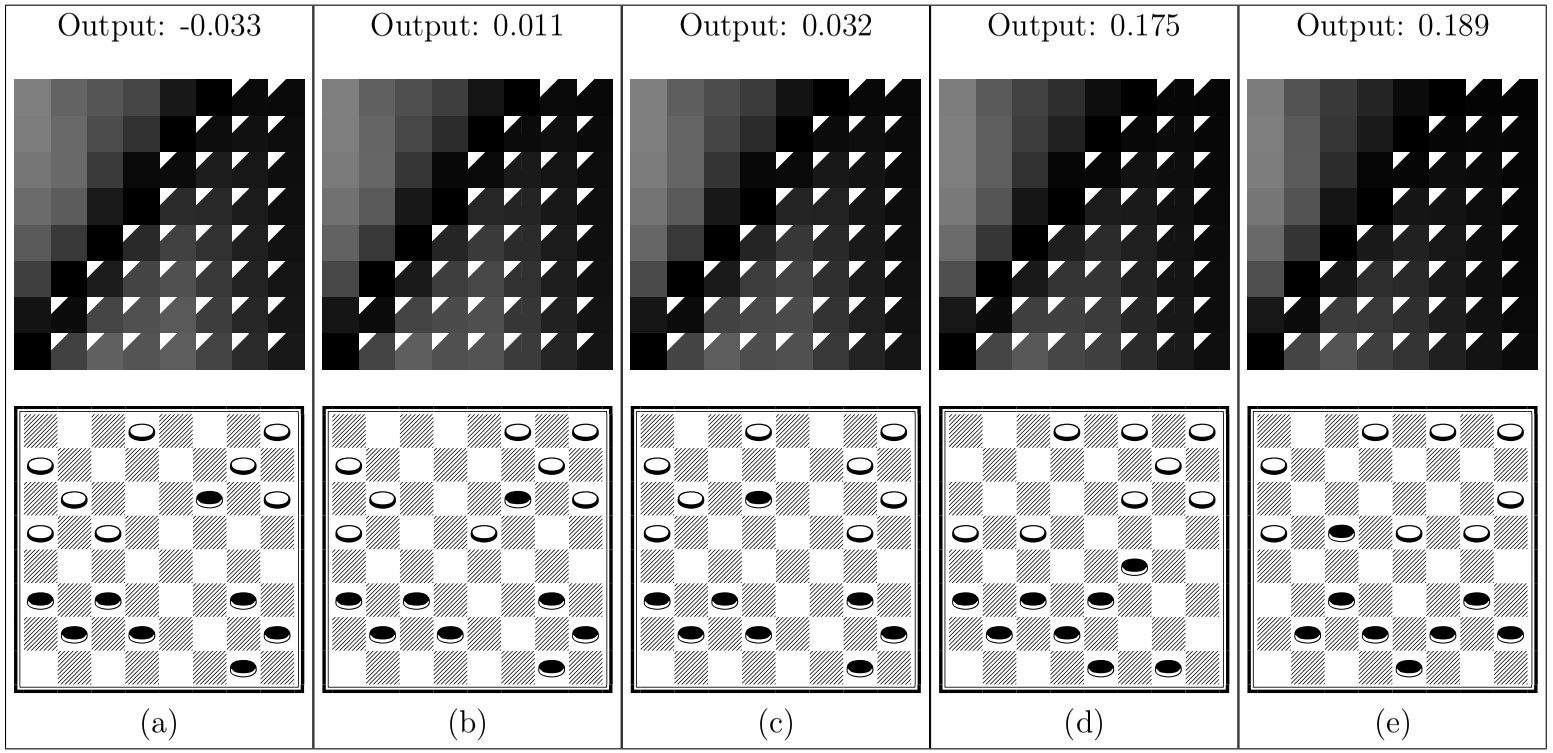


Figure 4.12: **Board positions and associated hidden layer activation patterns encountered by alpha-beta search for the general solution shown in figure 4.8c.** In (a), the breakaway black piece has no chance to escape, yielding a low evaluation. In evaluation (b), the piece still has no escape but is unable to be taken directly, producing a slightly higher score. Improving the situation slightly again (c), the piece is not in immediate danger but is too far up the board to be defended. The situation is best in (e) because, while black's piece is far up the board, it is backed up by additional pieces nearby.

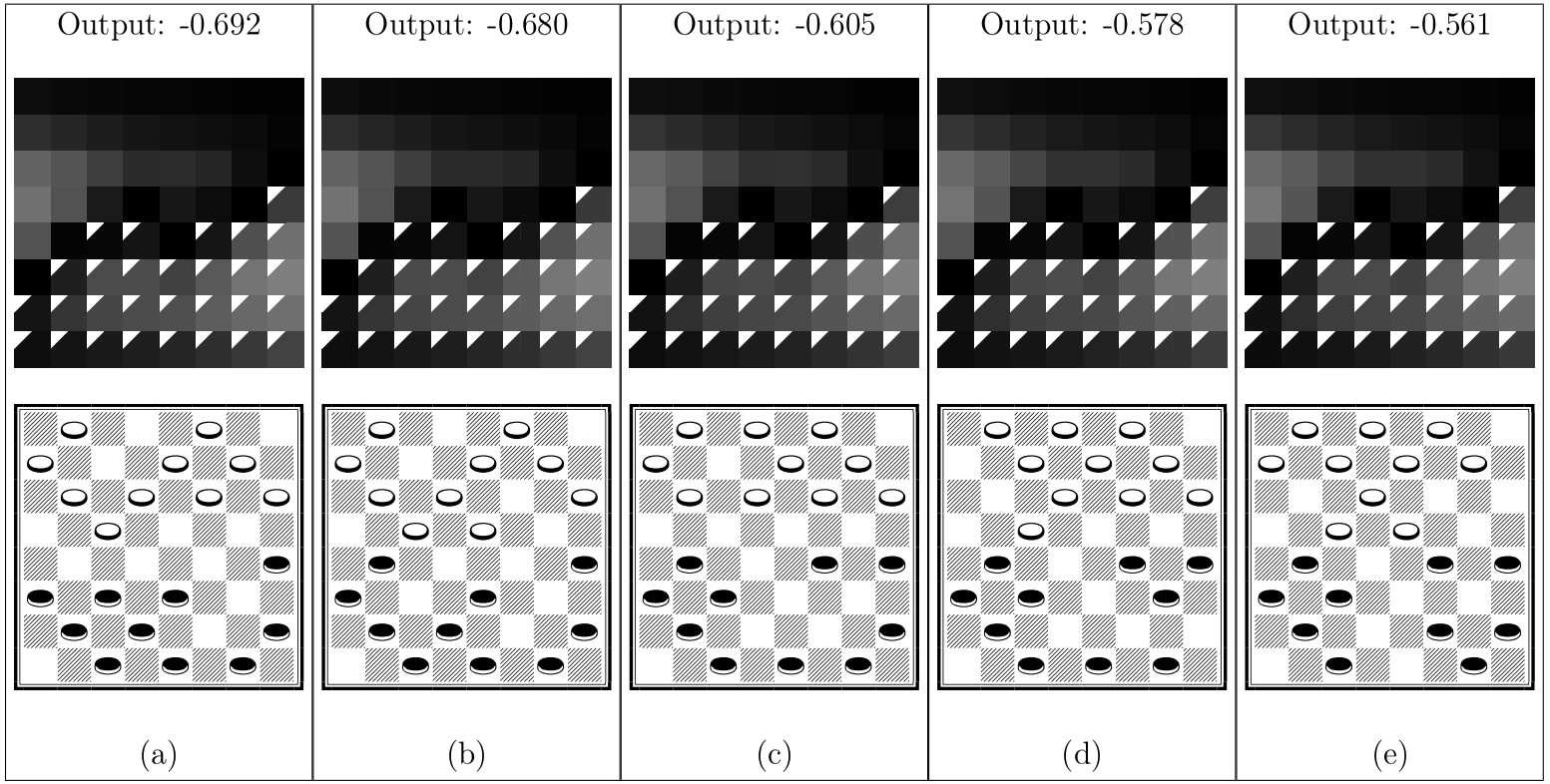


Figure 4.13: **Board positions and associated hidden layer activation patterns encountered by alpha-beta search for the general solution shown in figure 4.8d.** In the lowest scoring evaluation (a), only a single black piece is near the opponent's side of the board. There is an additional such black piece in (b), and three such black pieces in (c). In (d), the three black pieces are better defended, and in evaluation (e), white will be forced to take and will be less one piece in the center as a result. Thus this substrate favors an aggressive stance.

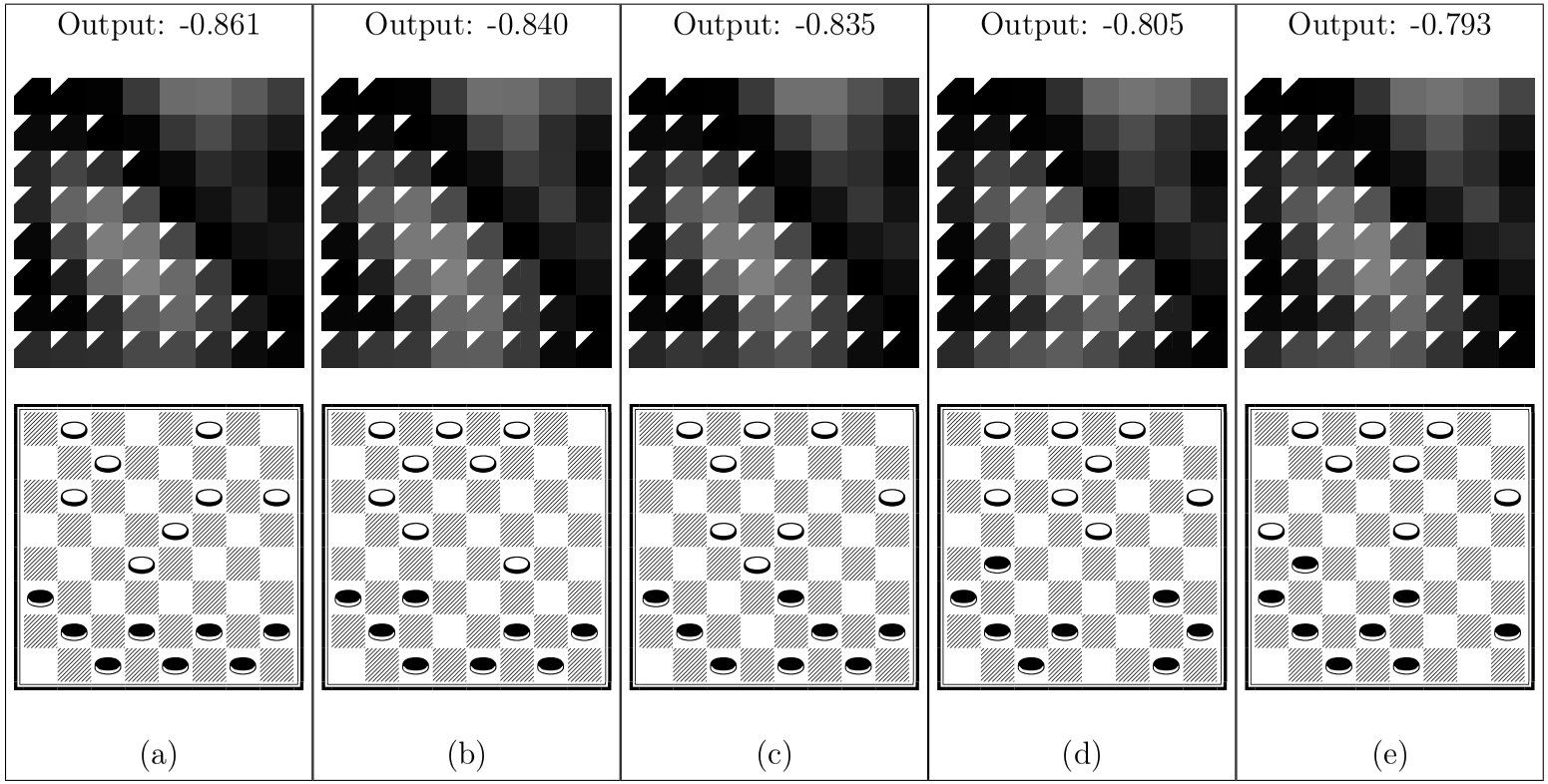


Figure 4.14: **Board positions and associated hidden layer activation patterns encountered by alpha-beta search for the less general solution shown in figure 4.9a.** In (a), there are several white pieces in the center. The black pieces creep forward in (b) and (c). In (d), white pieces do not have control of the center, and in (e), white has even less material in the center. However, this less general solution considers (e) a good move even though the white piece will double-jump on its next turn.

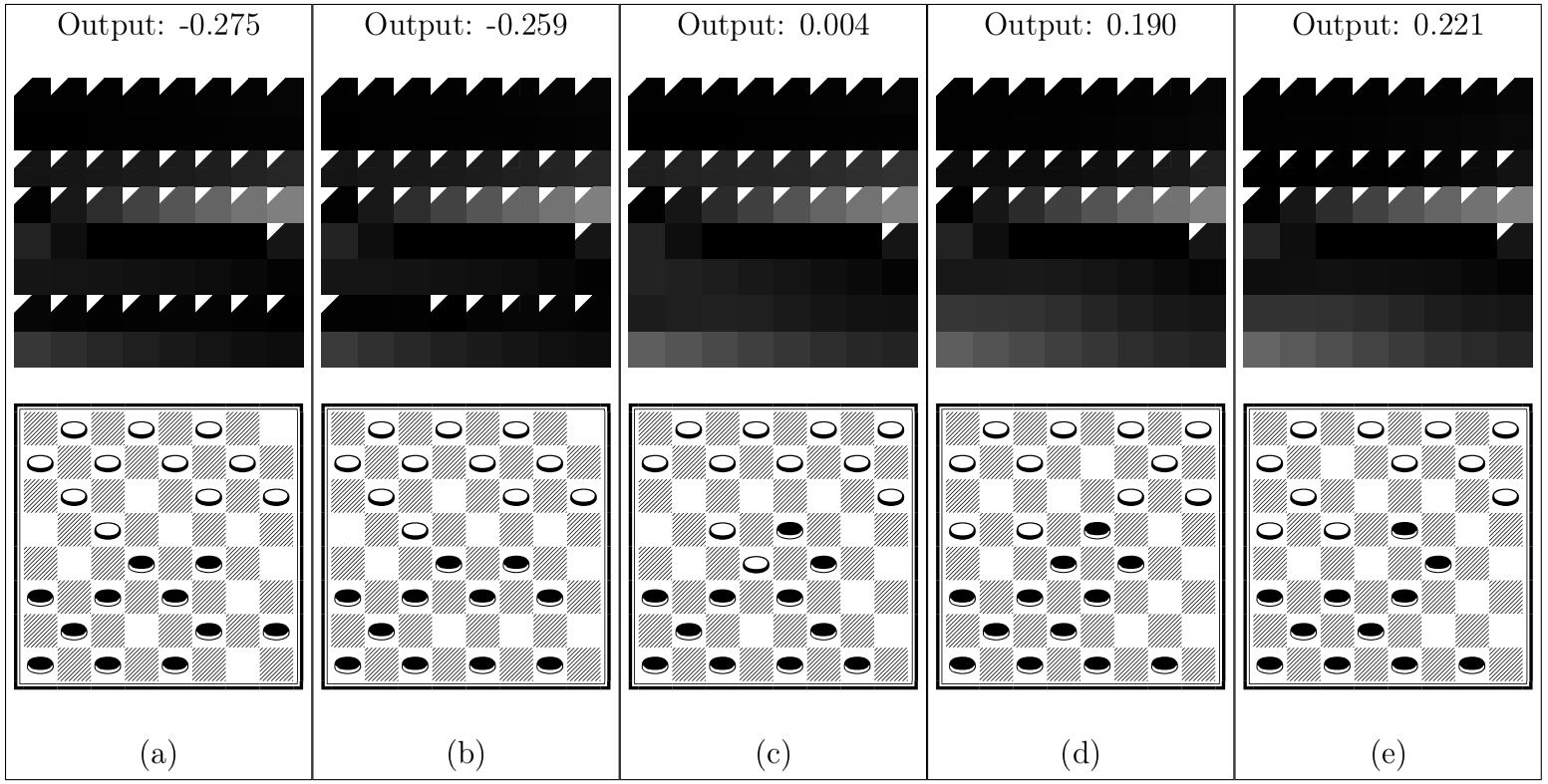


Figure 4.15: **Board positions and associated hidden layer activation patterns encountered by alpha-beta search for less general solution shown in figure 4.9b.** Moving from left (a) to right (e), black pieces assume more control of the center. However, this less general solution rates (d) highly even though the white piece in the middle will double-jump the center black pieces on the next turn.

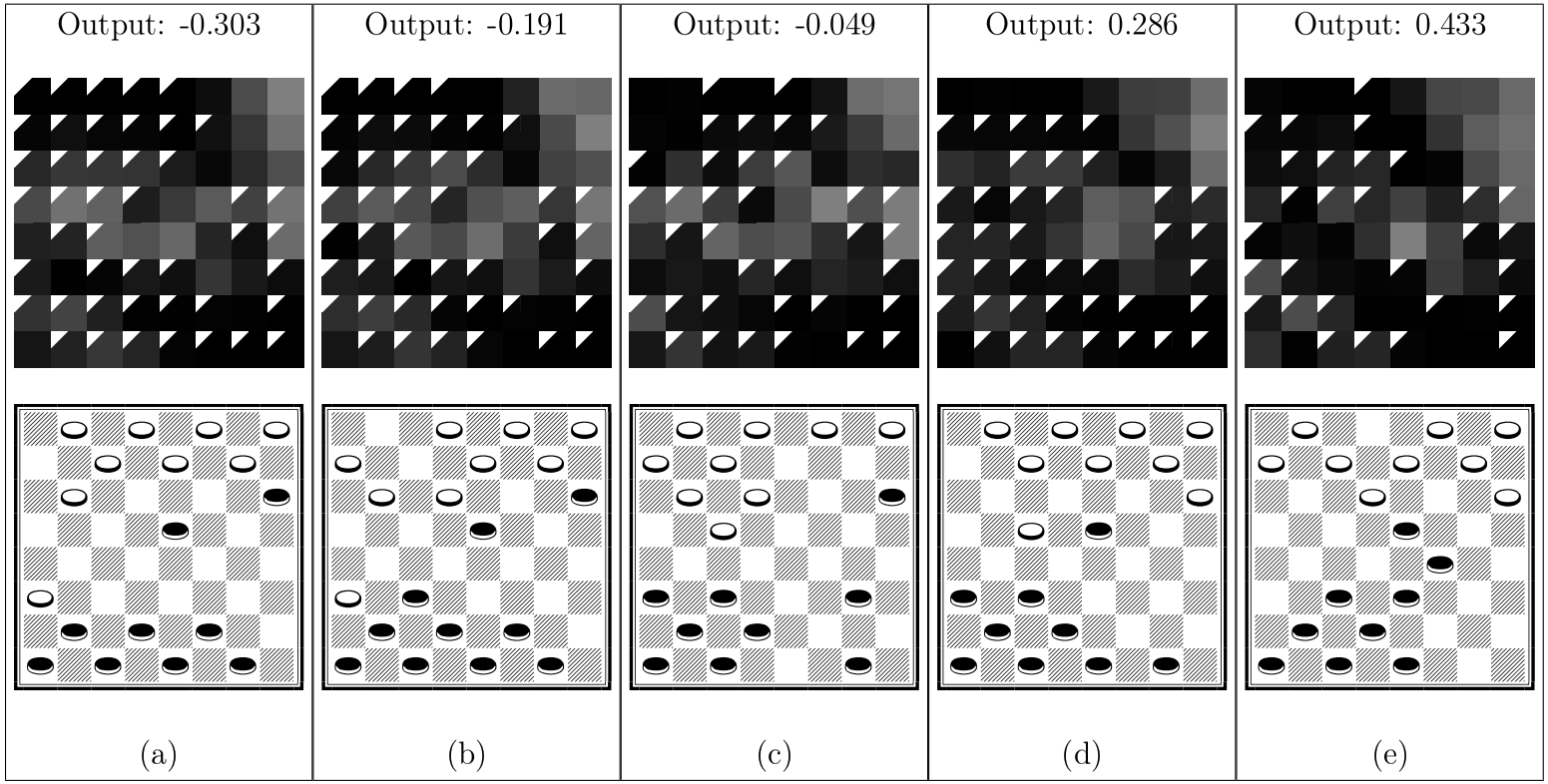


Figure 4.16: **Board positions and associated hidden layer activation patterns encountered by alpha-beta search for the less general solution shown in figure 4.9c.** In (a), two black pieces are far up the board, leaving little control of the center. As the evaluations improve in (b) through (e), black gains a stronger foothold on the center of the board and better support for pieces in white's territory. However, this less general solution favors (d) even though black's control of the center is prone to attack from white.

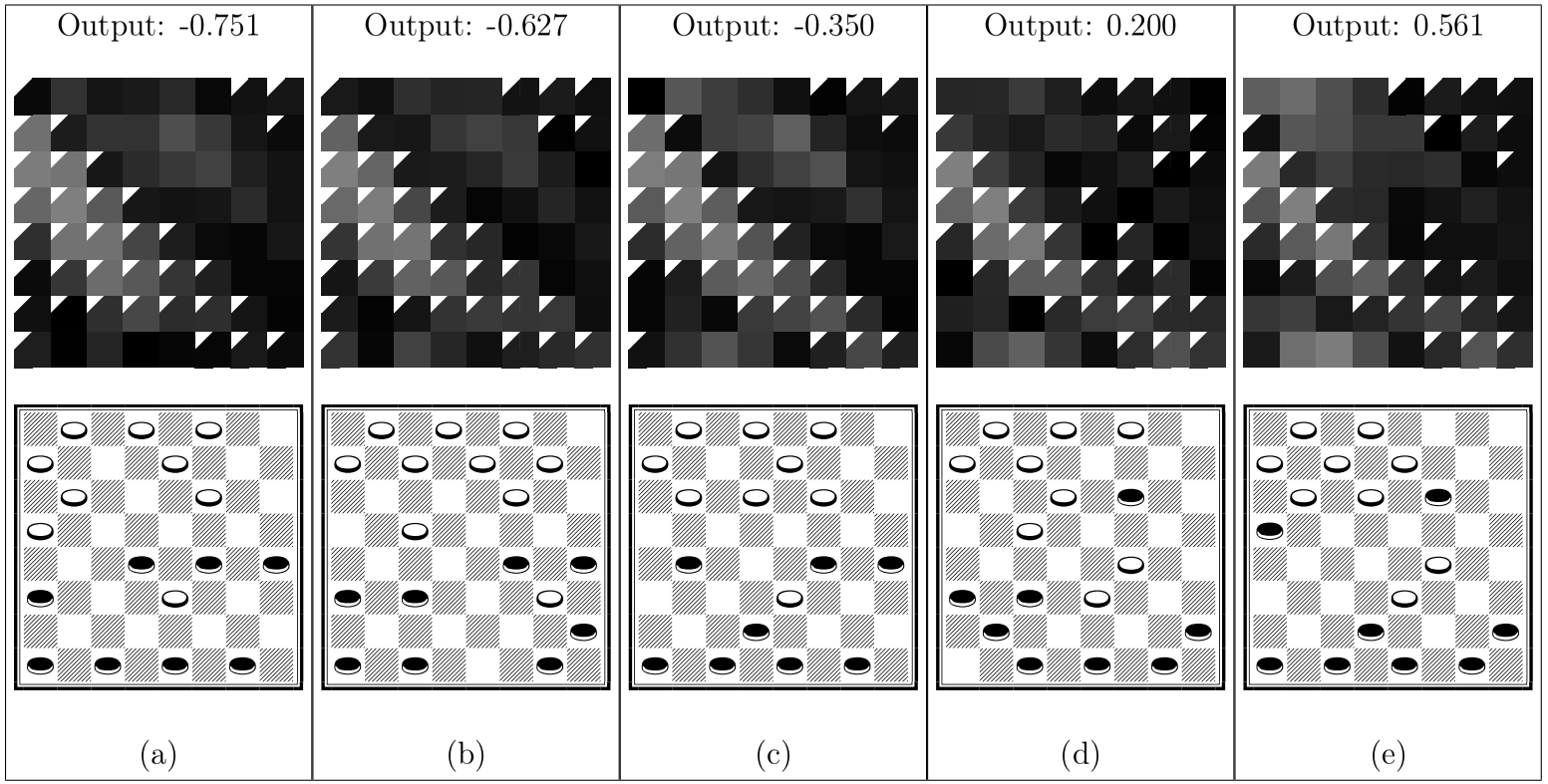


Figure 4.17: **Board positions and associated hidden layer activation patterns encountered by alpha-beta search for the less general solution shown in figure 4.9d.** Three black pieces in the center of (a) are hard to defend (i.e. they are spread out). Their position improves in (b). In (c), black also has a full back rank (i.e. all of the pieces on the back row are still in their starting configuration). One black piece is far up the board in (d), and in (e), black has both a piece far up the board and a full back rank. However, while this less general solution highly rewards (e), white is forced to capture black's most forward piece on the next turn.

rewards black for advancing up the board, but does not account for the fact that white will capture the leading piece on the next turn. The fact that solutions that do not have a smooth geometry make such mistakes despite defeating the deterministic heuristic further suggests that generality is linked to smooth geometry.

The analysis in this section shows what it *means* to learn from geometry. In effect, learning from geometry means being able to correlate topographic maps to the geometry of the world. This ability affords smooth regular connectivity patterns, which this section showed are often correlated to the more general checkers players. The next section explores the deeper implications of this discovery and what it means for artificial evolution in general.

4.4 IMPLICATIONS

A major difference between traditional multilayer perceptrons [MLPs; MRH86] and biological brains is that real brains profusely exploit topographic relationships [Spo02]. Some artificial neural models such as self-organizing maps [SOMS; Koh81] and cortical models [BKM02, BM06, Swi96] exhibit geometric structure, but the geometry of these models is either defined a priori or acquired through unsupervised learning, while HyperNEAT learns from geometry in a semi-supervised manner. A key contribution of this chapter is to show that it is possible for an evolutionary algorithm to actually evolve its own topographic maps

that are domain-appropriate. This development is intriguing because it means that neuroevolution algorithms can now produce structures more reminiscent of biological brains.

Furthermore, an important result is to show *why* evolving such structures is advantageous. In particular, at least in checkers, visualizing artificially evolved influence maps and receptive fields suggests an intimate connection between generalization and geometry. The connectivity patterns that exhibit smoothness, a qualitative assessment of the gradient across the hypercube of connection weights, were shown to be correlated to generalization, a quantitative assessment of the substrate against new opponents.

HyperNEAT is biased towards creating general players because the low complexity of the initial population of CPPNs tends to start evolution with simple, smooth geometries. However, it is not guaranteed to produce general results; several runs yielded less general solutions. Nevertheless, it is important to note that even these less general solutions still generalize significantly better than NEAT-EI on average, suggesting that NEAT-EI cannot easily encode the same smooth regularities that are demonstrated in HyperNEAT. Even worse, no runs of regular NEAT or FT-NEAT were able to defeat the *deterministic* heuristic in training, illustrating the necessity of capturing at least some geometric regularities, whether through an indirect encoding such as HyperNEAT, or through a engineered topology such as NEAT-EI. While engineering geometry into the network connectivity (as with NEAT-EI) provides NEAT a necessary advantage, it is not able to outperform HyperNEAT's ability to learn from geometry.

It is important to note that HyperNEAT solutions generalize significantly better than NEAT-EI solutions even though both methods trained against (and eventually defeated) the *same* heuristic in training. This difference is explained by HyperNEAT's indirect encoding: Because HyperNEAT CPPNs initially are much smaller than NEAT-EI genomes, they are biased towards representing substrates that are highly regular, but also successful in the domain of checkers. Because the direct encoding must learn each link weight individually, it searches through a comparatively high-dimensional space of neural networks, while the indirect encoding searches through a compressed (and hence lower-dimensional) space of solutions by leveraging its more powerful representation. HyperNEAT's representation naturally describes the geometric regularities of the problem domain. This capability helps in checkers because the domain (like many others) is inherently geometric. For example, the same rules generally apply to each piece at any position. Thus, the domain of checkers helps to illustrate the advantage of an indirect encoding based on geometry, such as in HyperNEAT.

However, the scope of domains that are inherently geometric is not limited to checkers and other board games. For example, visual discrimination [GS07a, SDG09] and robot control [DS07, DS08, SDG09, COP08a, CBO09a, COP09] domains can also benefit from indirect encoding through geometry. The inspirations for such domains are the vision and control systems of the human brain. In fact, topographic maps, which often have a geometry isomorphic to the external environment, are studied in the context of biological brains [CK04, Spo02, GC02, Chu86, KSJ00]. For example, the somatotopic representation of the

human body in the brain exhibits a similar geometry to the body itself [NYG98, YGS93]. Interestingly, ANNs evolved with HyperNEAT have receptive fields and influence maps that can be visualized much like such topographic maps in biological brains. Thus topographic maps in ANNs evolved by HyperNEAT are reminiscent of their more sophisticated biological counterparts, suggesting the start of an intriguing new direction of research in artificially-evolved substrates. In addition, not only do such maps arise, but their analysis helped to establish a connection between the smoothness and regularity of the geometry of such a map and its generalization. As a result, the surprising insight is that a qualitative assessment of evolved topographic maps translates directly to quantitative performance results in the generalization test.

4.5 SUMMARY

This chapter argued that representing evolved ANNs as indirect functions of their geometry evolves structures that are closer to structures seen in biological brains than those evolved by prior NE approaches. In addition, such a method is able to exploit the underlying geometric regularities in a problem to quickly find elegant solutions to complex problems, provided that geometry plays a role in the problem domain.

The role of geometry was shown to be potentially useful to machine learning performance in the domain of checkers. Regular NEAT and FT-NEAT were not able to defeat the

deterministic heuristic in a single run of training, while NEAT-EI and HyperNEAT were able to defeat the heuristic in all 20 runs. HyperNEAT was able to find solutions relatively quickly by searching through the low-dimensional space of CPPNs, while NEAT-EI took significantly longer, searching through the high-dimensional space of ANNs. In addition, solutions produced by HyperNEAT generalized significantly better than solutions produced by NEAT-EI, suggesting a link between HyperNEAT's perception of geometry and generality.

This link was confirmed through a visual study of general and less general HyperNEAT solutions and their performance in training. A correlation was drawn between the smoothness and continuity of connectivity patterns across the layers of solutions and their generalization performance, suggesting that general solutions encode ANNs that are smooth and regular, while less general solutions encode ANNs that are jagged and discontinuous. The CPPNs that encoded jagged ANNs were specialized for the specific games of checkers seen in training, while the CPPNs that encoded smooth ANNs were more general checkers players.

These results suggest that NE methods should ideally both see the geometry of the domain, and be able to encode and represent geometry in a way that creates smooth and regular ANNs. In this way, the ANNs produced by NE can more closely resemble neural networks seen in natural brains.

Another advantage of learning from geometry is simply that it removes the need for humans to decide how to engineer the representation. Thus, the primary lesson of this study is that machine learning should begin to integrate a capacity to learn geometric correlations and regularities in the task domain into its algorithms by providing them explicit access to

the domain geometry. The reward will be significantly more general solutions to real world problems. In the next two chapters, this approach is extended into a mature methodology for machine learning based on leveraging geometric information more effectively.

CHAPTER 5

SUBSTRATE EXTRAPOLATION

A major problem for neuroevolution is that the number of evaluations is related to the number of connections in the network being evolved [SM02]. Training a network with ten million connections can require significantly more evaluations than training one with one hundred. In HyperNEAT, it is possible to query the *same* CPPN at varying resolutions to create larger ANNs [SDG09] (figure 3.4). Thus, a promising potential approach to expanding network size is to learn basic concepts on a small ANN, increase the ANN resolution, and then continue learning more advanced concepts at the higher resolution (figure 5.1). This approach is designed to allow early, rapid learning of fundamental concepts.

5.1 CONTINUOUS VS. DISCRETE SUBSTRATE EXTRAPOLATION

There are two ways to scale inputs to an ANN that represent a geometric space. The first is to sample the input at a higher resolution. This form of scaling, called *continuous substrate extrapolation*, preserves the geometric relationships between locations on the input signal (figure 5.2a). The two images, while different resolutions, exist in the same geometric area.

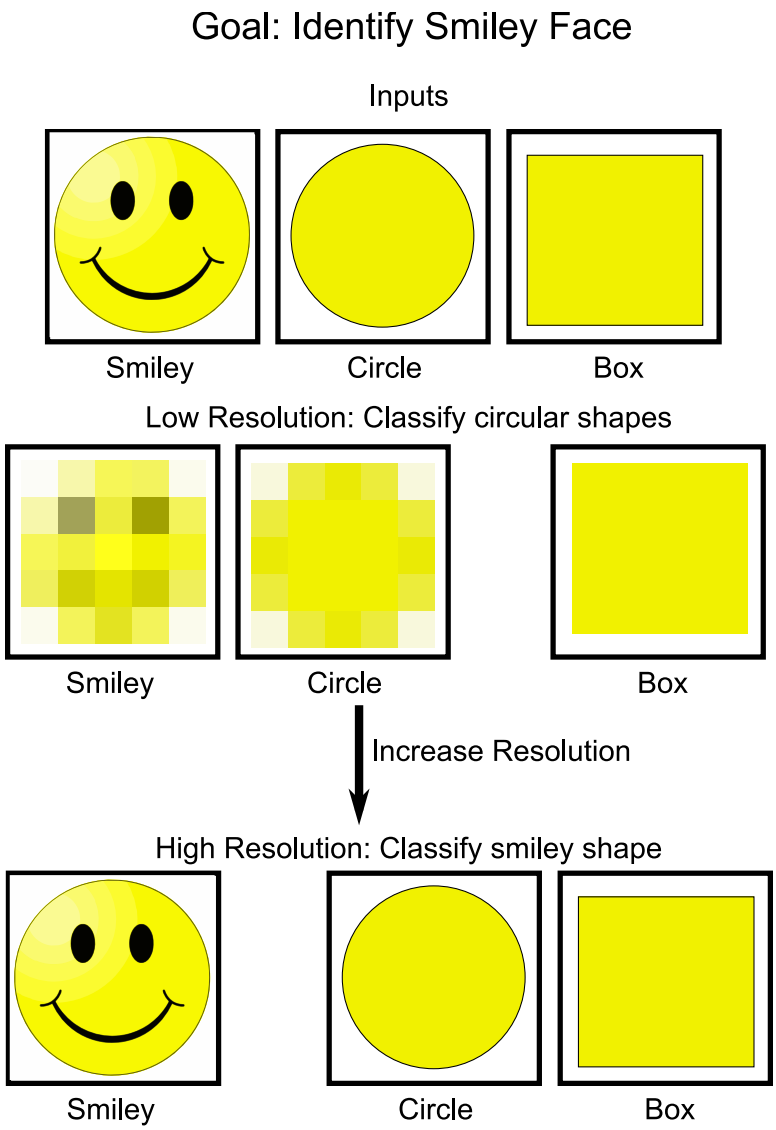


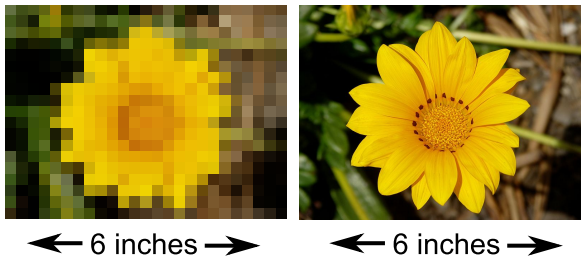
Figure 5.1: **Substrate Extrapolation.** The goal of the system shown above is to differentiate between the smiley-face picture and the other two shapes. At a low resolution, it is able to recognize circular shapes, but it lacks sufficient resolution to easily differentiate the smiley-face and the circle shapes. However, after increasing to a higher resolution, it is able to see the difference between the circle and smiley face and thereby learn at this new, higher resolution.

That is, a specific location in the image does not change its *meaning* even if the resolution of the image changes. As a result, the scaling changes *only* the distance between two adjacent pixels. Because CPPN inputs are by convention limited to a domain of $[-1, 1]$, the CPPN effectively normalizes the width and height of the image regardless of resolution, and can thereby extrapolate the ANN to handle this form of scaling naturally.

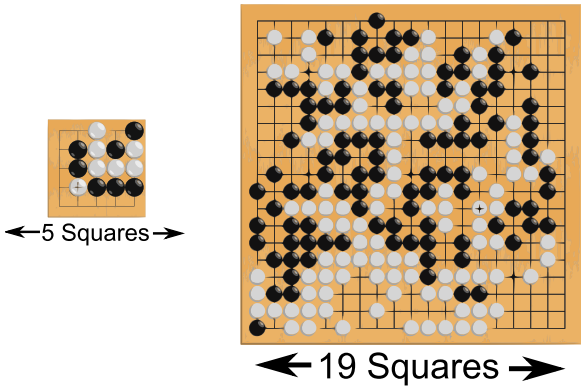
While this method can be effective in visual tasks, some domains do not lend themselves to this form of scaling. For example, if the resolution of the Go board in figure 5.2b is increased, the size of the domain *itself* is increased, as opposed to in the prior example, wherein it simply becomes more detailed. In such *discrete substrate extrapolation*, the size of a meaningful unit of information does not change as the resolution increases. As a result, a new method must be designed to handle this form of scaling.

5.2 DISCRETE SUBSTRATE EXTRAPOLATION IMPLEMENTATION

The problem in discrete extrapolation is that the range of the input domain changes as the scale increases (figure 5.2b). To address this phenomenon, it is necessary to first decide on the maximum resolution of the system. For example, in the game of Go in which board size can vary, this maximum resolution is 19×19, the size of the largest tournament Go board. The next step is to calculate the distance between two adjacent cells at this resolution. Because each input to the CPPN ranges from -1 to 1, the Go board must be rescaled to fit



(a) Continuous Extrapolation



(b) Discrete Extrapolation

Figure 5.2: **Continuous Versus Discrete Extrapolation.** In continuous substrate extrapolation (a), the bounds of the geometry do not change as the scale increases. In this case, the networks scale naturally with the domain. Note that the relative area of a single pixel decreases in continuous extrapolation. In discrete extrapolation (b), the relative area of a single square stays the same, but the overall geometry is expanded outward. In this case, special care is needed to ensure that the network scales appropriately with the domain.

this new range. Thus, the Go board position at index 0 maps to -1 and the position at index 18 maps to 1. The equation of a line, $y = y1 + \frac{(y2-y1)}{(x2-x1)} * (x - x1)$, fits the remaining indices to their appropriate CPPN input values. In the case of a 19×19 Go board, substituting the two points into the line equation gives: $y = -1 + \frac{2x}{18}$. Because the board index 1 (i.e. the second square in a row or column) maps to the CPPN input $\frac{-16}{18}$ and board index 0 maps to -1, the distance between two adjacent cells in the Go board is $\frac{2}{18}$.

Increasing the resolution of the substrate during evolution is an effective method to allow holistic complexification. This idea is similar to incremental evolution [GM97]: HyperNEAT can learn a low resolution task in a low dimensional space, and then increase resolution, continuing to evolve at a higher resolution. The next two chapters investigate both continuous and discrete substrate extrapolation.

CHAPTER 6

CONTINUOUS SUBSTRATE EXTRAPOLATION CASE STUDY: VISUAL DISCRIMINATION

A visual recognition problem is chosen to demonstrate HyperNEAT's ability to perform continuous substrate extrapolation. The experiment in this chapter is also published by Gauci and Stanley [GS07a] and Stanley et al. [SDG09].

6.1 Experiment

Vision is well-suited to testing learning methods on high-dimensional input. Natural vision also has the intriguing property that the same stimulus can be recognized equivalently at different locations in the visual field. For example, identical line-orientation detectors are spread throughout the primary visual cortex [CK04]. Thus there are clear regularities among the local connectivity patterns that govern such detection. A repeating motif likely underlies the general capability to perform similar operations at different locations in the visual field.

Therefore, a simple visual discrimination task is used to demonstrate HyperNEAT's capabilities. The task is to distinguish a large object from a small object in a two-dimensional

visual field. Because the same principle determines the difference between small and large objects regardless of their location in the retina, this task is well suited to testing the ability of HyperNEAT to discover and exploit regularities. It is important to note that this task may be solved more effectively in a scale-free manner with algorithms such as SIFT [Low04] or through the use of gaussian pyramids [BA83], but the goal of this experiment is to provide a benchmark for geometric indirect encodings and thereby assess the effectiveness of HyperNEAT to scale up with the domain.

The solution substrate is configured as a state-space sandwich (figure 3.3) that includes two sheets: (1) The *visual field* is a two-dimensional array of sensors that are either on or off (i.e. black or white). (2) The *target field* is an equivalent two-dimensional array of outputs that are activated at variable intensity between zero and one. In a single trial, two objects, represented as black squares, are situated in the visual field at different locations. One object is three times as wide and tall as the other (figure 6.1). The goal is to locate the center of the largest object in the visual field. The target field specifies this location as the node with the highest level of activation. Thus, HyperNEAT must discover a connectivity pattern between the visual field and target field that causes the correct node to become most active regardless of the locations of the objects.

An important aspect of this task is that it utilizes a large number of inputs, many of which must be considered simultaneously. To solve it, the system needs to discover the general principle that underlies detecting relative sizes of objects. The right idea is to strongly connect individual input nodes in the visual field to *several* adjacent nodes around

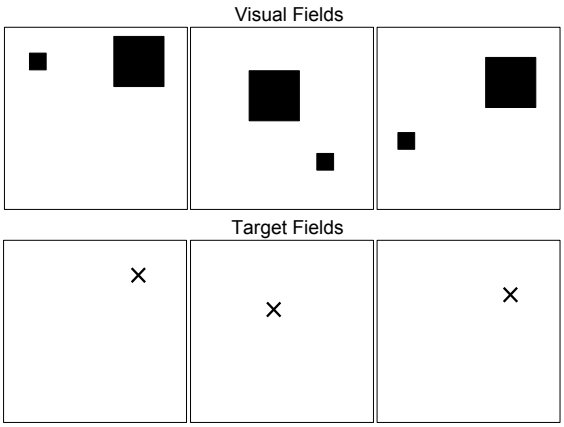


Figure 6.1: **The Visual Discrimination Task.** The task is to identify the center of the larger box. Example visual field activation patterns (top) and the corresponding correct target fields (bottom) are depicted. The "X" in each target field denotes the point of highest activation, which is how the ANN specifies the location of the center of the larger box. This task effectively tests HyperNEAT's ability to discover regularity because the same principle differentiates the larger box from the smaller one regardless of where the boxes appear on the input field.

the corresponding location in the output field, thereby causing outputs to accumulate more activation the more adjacent loci are feeding into them. Thus, the solution can exploit the geometric concept of *locality*, which is inherent in the arrangement of the two-dimensional grid. Only a representation that takes into account substrate geometry can exploit such a concept. Furthermore, an ideal encoding should develop a representation of the concept that is independent of the visual field resolution. Because the correct motif repeats across the substrate, in principle a connective CPPN can discover the general concept only once and cause it to be repeated across the grid at any resolution. As a result, such a solution can scale as the resolution inside the visual field is increased through continuous substrate extrapolation, even without further evolution.

6.2 EVOLUTION AND PERFORMANCE ANALYSIS

The field coordinates range between $[-1, 1]$ in the x and y dimensions. However, the resolution within this range, i.e. the node density, can be varied. During evolution, the resolution of each field is fixed at 11×11. Thus the connective CPPN must learn to correctly connect a visual field of 121 inputs to a target field of 121 outputs, a total of 14,641 potential connection strengths. If the magnitude of the CPPN's output for a particular query is less than or equal to 0.2 then the connection is not expressed in the substrate. If it is greater than 0.2, then the number is scaled to a magnitude between zero and three. The sign is preserved, so negative output values correspond to negative connection weights.

During evolution, each individual in the population is evaluated for its ability to find the center of the bigger object. If the connectivity is not highly accurate, it is likely the substrate will often incorrectly choose the small object over the large one. Each individual evaluation thus includes 75 *trials*, where each trial places the two objects at different locations. The trials are organized as follows. The small object appears at 25 uniformly distributed locations such that it is always completely within the visual field. For each of these 25 locations, the larger object is placed five units to the right, down, and diagonally, once per trial. The large object wraps around to the other side of the field when it hits the border. Note that the large object is either completely on one side or the other, and is not broken across the border. If the larger object is not completely within the visual field, it is moved the smallest distance possible that places it fully in view. Because of wrapping, this method of evaluation

tests cases where the small object is on all possible sides of the large object. Thus many relative positions are tested for a total number of 75 trials on the 11 by 11 substrate for each evaluation during evolution.

Within each trial, the substrate is activated over the entire visual field. The unit with the highest activation in the target field is interpreted as the substrate's selection. Fitness is calculated from the sum of the squared distances between the target and the point of highest activation over all 75 trials. This fitness function rewards generalization and provides a smooth gradient for solutions that are close but not perfect.

To demonstrate HyperNEAT's ability to effectively discover the task's underlying regularity, two approaches are compared.

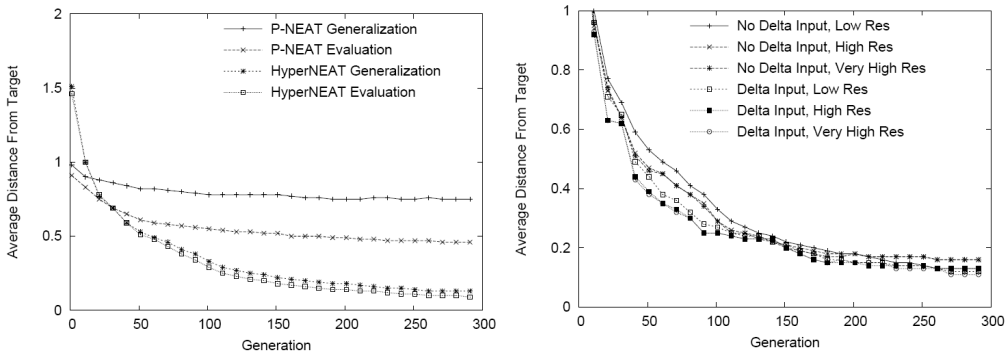
- **HyperNEAT:** HyperNEAT evolves a connective CPPN that generates a substrate to solve the problem (Chapter 3).
- **Perceptron Neat (P-NEAT):** P-NEAT is a reduced version of NEAT that evolves perceptrons (i.e. not CPPNs). ANNs with 121 nodes and 14,641 (121×121) links are evolved without structure-adding mutations. P-NEAT is run with the same settings as HyperNEAT because both are being applied to the same problem. Because P-NEAT must explicitly encode the value of each connection in the genotype, it cannot encode underlying regularities and must discover each part of the solution connectivity independently.

This comparison is designed to show how HyperNEAT makes it possible to optimize very high-dimensional structures, which is difficult for directly-encoded methods. HyperNEAT is then tested for its ability to scale solutions to higher resolutions *without* further evolution, which is *impossible* with direct encodings such as P-NEAT. The parameters for this experiment are provided in Appendix A.

6.3 RESULTS

The primary performance measure in this section is the *average distance from target* of the target field's chosen position. This average is calculated for each generation champion across all its trials (i.e. object placements in the visual field). Reported results were averaged over 20 runs. Better solutions choose positions closer to the target. To understand the distance measure, note that the width and height of the substrate are 2.0 regardless of substrate resolution.

HyperNEAT and P-NEAT were compared to quantify the advantage provided by generative encoding on this task. During evolution, both HyperNEAT and P-NEAT improved over the course of a run. Figure 6.2a shows the performance of both methods on *evaluation* trials from evolution (i.e. a subset of all possible positions) and on a generalization test that averaged performance over *every* possible valid pair of positions on the board. An input



(a) P-NEAT and HyperNEAT Generalization

(b) HyperNEAT Generation Champion Scaling

Figure 6.2: **Generalization and Scaling.** The graphs show performance curves over 300 generations averaged over 20 runs each. (a) P-NEAT is compared to HyperNEAT on both evaluation and generalization. (b) HyperNEAT generation champions with and without delta inputs are evaluated for their performance on 11×11, 33×33, and 55×55 substrate resolutions. The results show the HyperNEAT generalizes significantly better than P-NEAT ($p < 0.01$) and scales almost perfectly.

is considered valid if the smaller and the larger object are placed within the substrate and neither object overlaps the other.

The performance of both methods on the evaluation tests improved over the run. However, after generation 45, on average HyperNEAT found significantly more accurate solutions than P-NEAT ($p < 0.01$).

HyperNEAT learned to generalize from its training; the difference between the performance of HyperNEAT in generalization and evaluation is not significant past the first generation. Conversely, P-NEAT performed significantly worse in the generalization test after generation 51 ($p < 0.01$). This disparity in generalization reflects HyperNEAT's fundamental ability to learn the geometric concept underlying the task, which can be generalized across the substrate. P-NEAT can only discover each proper connection weight independently.

Therefore, P-NEAT has no way to extend its solution to positions in the substrate on which it was never evaluated. Furthermore, the search space of 14,641 dimensions (i.e. one for each connection) is too high-dimensional for P-NEAT to find good solutions while HyperNEAT discovers near-perfect (and often perfect) solutions on average.

6.4 SCALING PERFORMANCE

The best individuals of each generation, which were evaluated on 11×11 substrates, were later scaled through continuous substrate extrapolation with the *same CPPN* to resolutions of 33×33 and 55×55 by requerying the substrate at the higher resolutions without further evolution. These new resolutions cause the substrate size to expand dramatically. For 33×33 and 55×55 resolutions, the weights of over *one million* and *nine million* connections, respectively, must be optimized in the substrate, which would normally be an enormous optimization problem. On the other hand, the original 11×11 resolution on which HyperNEAT was trained contains only up to 14,641 connections. Thus, the number of connections increases by nearly three orders of magnitude. It is important to note that HyperNEAT is able to scale to these higher resolutions without any additional learning. In contrast, P-NEAT has no means to scale to a higher resolution and cannot even learn effectively at the lowest resolution.

When scaling, a potential problem is that if the same activation level is used to indicate positive stimulus as at lower resolutions, the total energy entering the substrate would in-

crease as the substrate resolution increases for the same images, leading to over-saturation of the target field. In contrast, in the real world, the number of photons that enter the eye is the same regardless of the density of photoreceptors. To account for this disparity, the input activation levels are scaled for larger substrate resolutions proportionally to the difference in unit cell size.

Two variants of HyperNEAT were tested for their ability to scale (figure 4.2b). The first evolved the traditional CPPN with inputs x_1, y_1, x_2, and y_2. The second evolved CPPNs with the additional *delta* inputs $(x_1 - x_2)$ and $(y_2 - y_1)$. The intuition behind the latter approach is that because distance is a crucial concept in this task, the extra inputs can provide a useful bias to the search.

While the deltas did perform significantly better on average between generations 38 and 70 ($p < 0.05$), the CPPNs without delta inputs were able to catch up and reach the same level of performance after generation 70. Thus, although applicable geometric coordinate frames provide a boost to evolution, HyperNEAT is ultimately powerful enough to discover the concept of distance on its own.

Most importantly, both variants were able to scale through continuous substrate extrapolation almost perfectly from 11×11 resolution substrate with up to 14,641 connections to a 55×55 resolution substrate with up to 9,150,625 connections, with no significant difference in performance after the second generation. This result is also significant because the higher-resolution substrates were tested on *all* valid object placements, which include

many positions that did not even exist on the lower-resolution substrate. Thus, remarkably, CPPNs found solutions that lose no abilities at higher resolution!

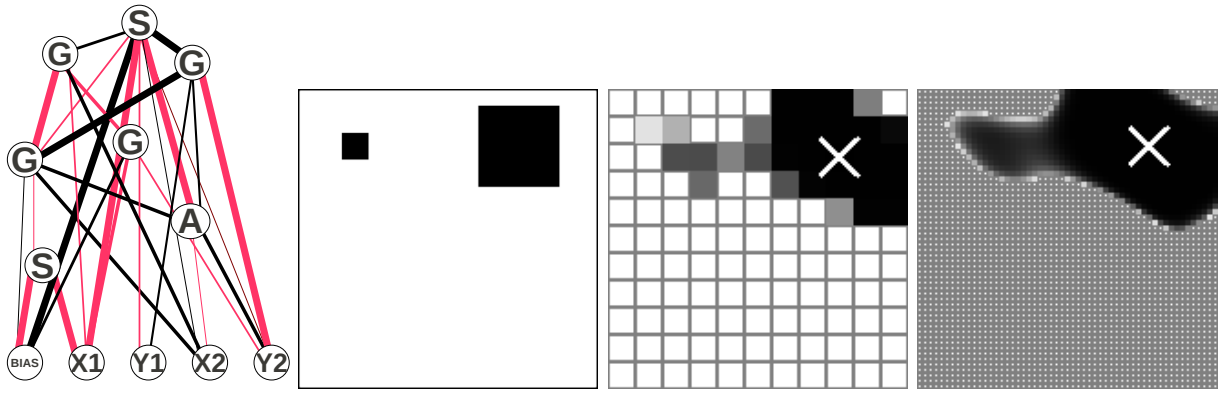
High-quality CPPNs at the 55×55 resolution contained on average *8.3 million connections* in their substrate and performed as well as their 11×11 counterparts. These substrates were the largest functional structures produced by evolutionary computation of which we were aware when they were first published [GS07a].

CPPN encoding is highly compact. If *good solutions* are those that achieve an average distance under 0.25, the average complexity of a good solution CPPN was only 24 connections. In comparison, at 11×11 resolution the average number of connections in the *substrate* was 12,827 out of the possible 14,641 connections. Thus the genotype is smaller than the evaluated phenotype on average by a factor of 534.

6.5 SUBSTRATE EXTRAPOLATION ANALYSIS

Examples of scaling are shown in figure 6.3, which shows how the activation pattern looks on different resolution substrates generated by the same CPPN. This motif is sufficiently robust that it works at variable resolutions i.e. the encoding is scalable.

Although HyperNEAT generates ANNs with millions of connections, such ANNs can run in real time on most modern hardware. Using a 2.0GHz processor, an eight million



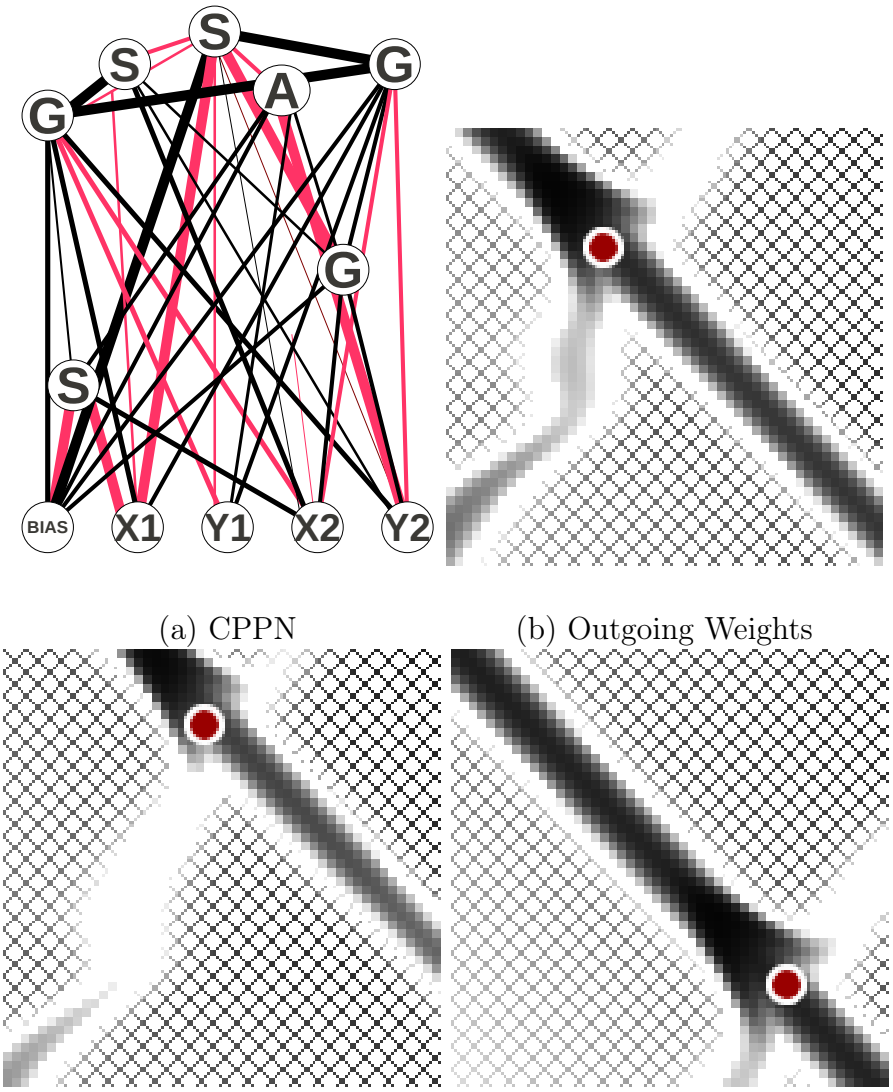
(a) CPPN (b) Input Pattern (c) 11×11 Output (d) 55×55 Output

Figure 6.3: **Activation Patterns of the Same Connective CPPN at Different Resolutions.** Activation patterns on the target field of a substrate generated by the CPPN in (a) from the input trial shown in (b) are displayed at resolution 11×11 in (c) and 55×55 in (d). Darker color signifies higher activation and the position of highest activation is marked with a white "X." The same 26-connection CPPN generates solutions at both resolutions, with 10,328 and 8,474,704 connections, respectively, demonstrating the ability of the solution to scale significantly.

connection network takes on average 3.2 minutes to create, but only 0.1 seconds to process a single trial.

6.6 REPEATING PATTERNS

To identify the largest object irrespective of both objects' locations, HyperNEAT must discover a fundamental connectivity motif originating from each input neuron and repeat it across the substrate. Figure 6.4 shows a *diagonal cross* connectivity motif originating from three different input neurons on the same substrate, which is generated by the connective CPPN in figure 6.4a.



(a) CPPN (b) Outgoing Weights

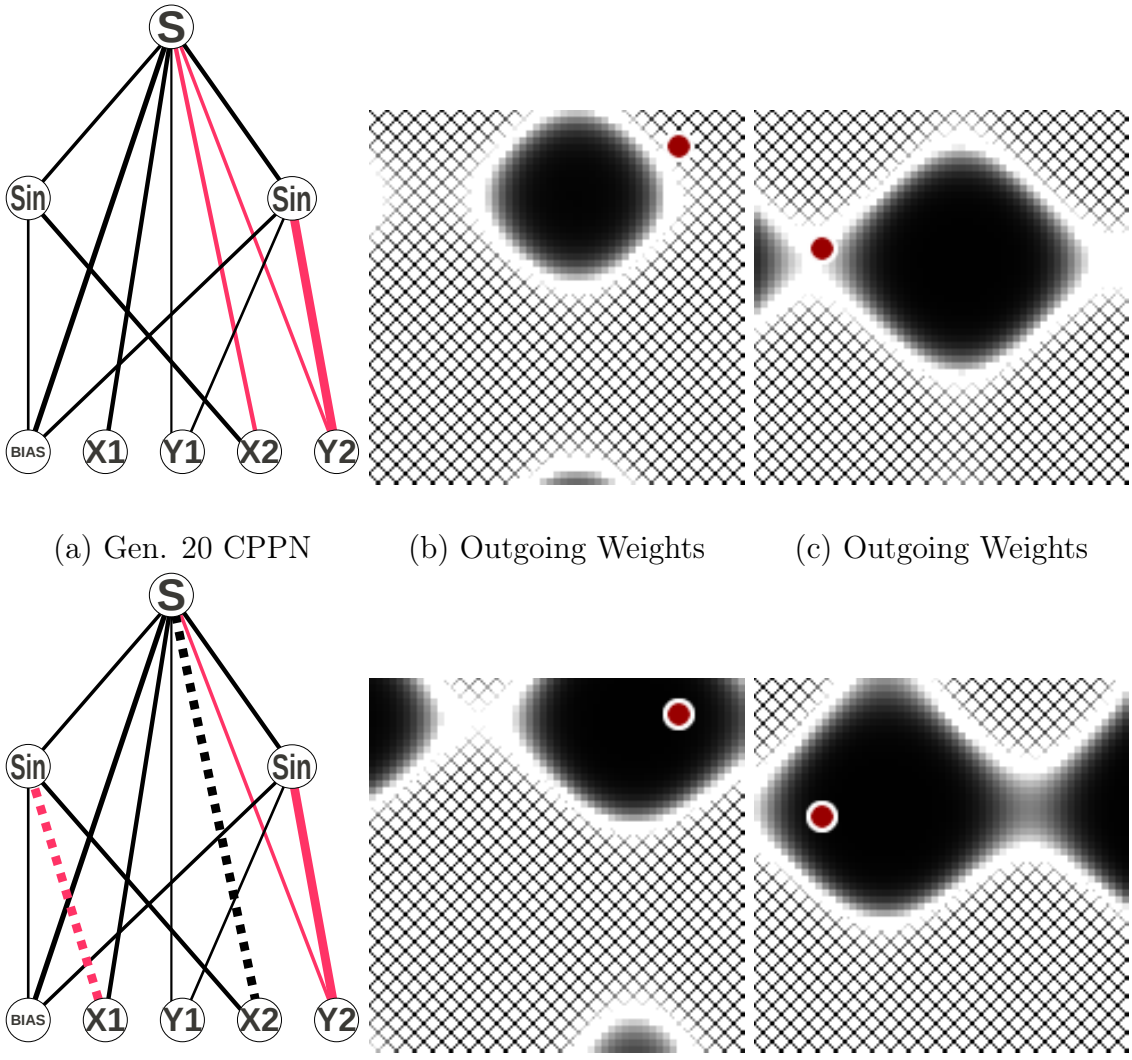
(c) Outgoing Weights (d) Outgoing Weights

Figure 6.4: **Connectivity Motifs of the Same Substrate at Different Locations.** The CPPN in (a) generates the motifs shown in (b–d), which represent outgoing connectivity patterns from a single node in the visual field, whose position is denoted by a small dot (i.e. each frame is a two-dimensional cross-section of the four-dimensional hypercube encoded by the CPPN). Note that these patterns, which each originate from only one node, differ from those in figure 6.3, which shows activation patterns from an entire trial with multiple simultaneous active nodes. The cross-diagonal hatch background represents areas of negative weight, while solid colors between white and black represent increasingly-high positive weights. The figure shows that the connective CPPN is able to repeat the same motif across the substrate.

HyperNEAT discovered several motifs that are all effective at this task. The most intuitive motif is the halo, which produces activation patterns such as those in figure 6.5. Halo motifs were discovered in eight of the 20 runs. Although they were less common, diagonal cross motifs evolved separately four times. The remaining eight runs produced a variety of different shapes that all work equally well. These results show that HyperNEAT creatively exploits the task geometry to find a variety of effective repeating patterns.

6.7 DISCOVERING REGULARITIES

Figure 6.5 shows a solution at two different generations in the same run, illustrating the unique process through which regularities in the solution are discovered. In generation 20 (figure 6.5a–c), the substrate produces halo connectivity patterns projecting from a single input node that are at the correct vertical position, but it has not yet learned the principle of horizontal locality. Four generations later (figure 6.5d–f), it augments this concept by adding a new connection from x_1 (i.e. its horizontal position) to an internal sine function (figure 6.5d). This change recalibrates the horizontal position of the halo center to be closer to the source node. This example explicitly demonstrates the process through which geometric relationships are discovered. In effect, the problem is recast from finding the correct weight of every connection originating from the visual field to a problem of finding the geometric concepts that underlie the solution.



(a) Gen. 20 CPPN (b) Outgoing Weights (c) Outgoing Weights

(d) Gen. 24 CPPN (e) Outgoing Weights (f) Outgoing Weights

Figure 6.5: **Discovering Regularities through CPPN Complexification.** CPPNs and their respective substrate output are depicted at generations 20 (a–c) and 24 (d–f). As in figure 6.4, the connectivity pattern originates in each case from the location of the small dot. The figure shows that the CPPN learned to horizontally calibrate the positions of positive-weighted connections in the substrate by discovering a new connection from x_1 and changing the sign of the connection from x_2. Both connections are highlighted in (d) as dotted lines. Thus, HyperNEAT learns high-level concepts rather than searching for the weight of individual connections in a massive ANN independently. The 13-connection CPPN in (d) produces the 8,644,480-connection substrate in (e) and (f).

6.8 SUMMARY

Like other direct encodings, P-NEAT can only improve by discovering each connection strength individually. Furthermore, P-NEAT solutions cannot generalize to locations outside its training corpus because it has no means to represent the general pattern of the solution. In contrast, HyperNEAT discovers a general connectivity concept that naturally covers locations outside its training set. Thus, HyperNEAT creates a novel means of generalization, through substrate geometry. That is, HyperNEAT *exploits the geometry of the problem* by discovering its underlying regularities.

Because the solution is represented conceptually, the same solution effectively scales to higher resolutions through continuous substrate extrapolation, which is a new capability for ANNs. Thus a working solution was possible to produce with over eight million connections, which is among the largest working ANNs ever produced through evolutionary computation. The real benefit of this capability however is that in more sophisticated domains, further evolution can be performed at the higher resolution, which is an opportunity exploited through discrete substrate extrapolation in the next chapter. The main conclusion is that connective CPPNs are a powerful new generative encoding that can compactly represent and scale up to large-scale ANNs with very few genes.

It is important to note that HyperNEAT is not restricted to state-space sandwich substrates. The method is sufficiently general to generate arbitrary two-dimensional or three-dimensional connectivity patterns, including hidden nodes (as demonstrated in Chapter 4)

and recurrence. Thus the concepts in this chapter apply to a broad range of high-resolution problems with potential geometric regularities. While the domain in this chapter demonstrated continuous substrate extrapolation (Section 5.1), the following chapter applies discrete substrate extrapolation (Section 5.2) to the domain of scalable Go.

CHAPTER 7

DISCRETE SUBSTRATE EXTRAPOLATION CASE STUDY: SCALABLE GO AGAINST SIMPLEPLAYER

The game of Go has attracted much attention from the artificial intelligence community. A key feature of Go is that humans begin to learn on a small board, and then incrementally learn advanced strategies on larger boards. While some machine learning methods can also scale the board, they generally only focus on a subset of the board at one time. Neuroevolution algorithms particularly struggle with scalable Go because they are often directly encoded (i.e. a single gene maps to a single connection in the network). Thus this chapter applies HyperNEAT to the problem of scalable Go. HyperNEAT can evolve a solution to 5×5 Go and then *extrapolate* that solution to 7×7 Go and continue evolution. The scalable method is demonstrated to learn faster and ultimately discover better strategies than the same method trained on 7×7 Go directly from the start.

7.1 MOTIVATION

The game of Go is challenging for artificial intelligence. Unlike games with low-branching search trees that can either be solved completely (e.g. connect four) or searched to a deep ply to great effect (e.g. checkers and chess), the branching factor and state space in Go render these traditional approaches intractable [BW95]. Go demands new search techniques to reduce the branching factor, and abstract representations that can consolidate the state space. Because such representations are often difficult to conceptualize, human-engineered heuristics are often ineffective. One promising alternative is machine learning, wherein techniques such as temporal difference learning or neuroevolution learn a value function from an abstract representation [SSM07, SS08, SM04b].

Yet even with such innovations, experienced human Go players can still consistently defeat the strongest of computer players without a handicap [Bot08]. One notable difference between human players and most machine learning-based approaches to Go is that the human player begins to learn Go on a small board [Sho03]. Humans can then *extrapolate* information learned on the smaller board to a larger board, thereby bootstrapping from it. Such extrapolation is challenging for machine learning algorithms, which often cannot transfer knowledge from one board size to another.

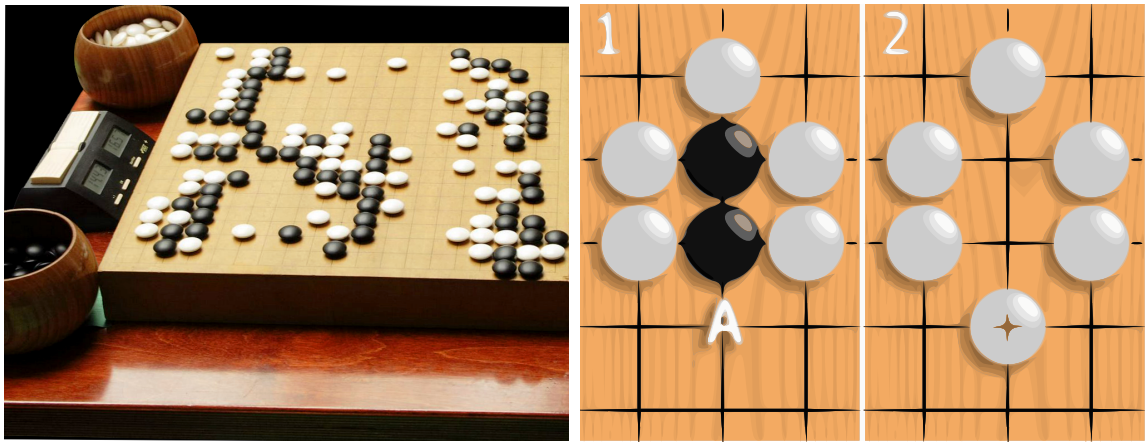
However, several notable exceptions exist that typically fall into one of two categories: (1) The first convert the Go board into a set of local features that are independent of the board size [SSM07]; (2) the second class of methods scan sections of the board and *remember*

notable positions and information [SM04b, SS08]. In both cases, the key is to view a small section of the Go board at one time. As a result, it is potentially difficult to develop holistic strategies. This challenge is especially problematic for Go, in which many tactics (e.g. ladders) depend on a holistic view of the board.

In this chapter, a new method of scaling based on discrete substrate extrapolation is presented that breaks from the aforementioned techniques, yet can still scale the board to new sizes and continue learning [GS10b]. HyperNEAT ultimately produces the ability to scale the Go board to new sizes *without changing the representation* and continue evolution. The result is that candidates evolved on 5×5 Go and then scaled and evolved further at 7×7 Go outperform candidates evolved solely on 7×7 Go without scaling. Thus the main contribution is to show that indirect encoding is a viable foundation for training scalable learners, and offers the unique potential to represent holistic solutions at variable sizes.

7.2 BASICS OF GO

Go is a two player board game [Bot08, Sho03]. The players take turns placing stones on an $n \times n$ grid. The standard board size is 19×19 (figure 7.1a), however, common board sizes also include 5×5, 9×9, and 13×13. The player's objective is to possess more stones on the board than the opponent at the end of the game. If a player is able to form a complete border around a group of the opponent's stones, the surrounded stones are removed from



(a) Go Board (b) Go Capture

Figure 7.1: **Rules of Go.** A typical Go board is a grid of intersection points (a). Although the board shown is 19×19, other board sizes are also playable. In Go (b), white places a token on the intersection marked "A", surrounding black's tokens, which are removed from the board. (Images are courtesy of Wikipedia CC3.0, GFDL 1.2.)

the board (figure 7.1b). The game ends when both players pass. At this point, both players agree that there are no beneficial moves for either player and the game is thus deadlocked in their view. The player with the most territory is then declared the winner. Territory is defined by the number of stones a player has plus any empty squares that are agreed by both players to be unobtainable by the opponent. A complete description of Go can be found in Botermans [Bot08] and Shotwell [Sho03].

Go is challenging for computers because there are many possible legal moves in most board positions [GS08b]. The average number of possible moves is referred to as the *branching factor*. The computation time increases exponentially with the branching factor. Unlike e.g. in chess, any unoccupied position on the board is a potential move for the player. Thus the branching factor is high. For example, in the opening move, there are $n \times n$ possible moves,

where n is the size (i.e. length) of the board. In e.g. 19×19 Go there are 361 possible first moves. This number contrasts significantly with chess, which has 10 possible first moves. On the other hand, in 5×5 Go, only 25 first moves are possible. Thus a learning method that can learn on a smaller board and then scale to a larger board can potentially learn significantly faster than a method that must learn from scratch on the larger board.

7.3 PRIOR WORK ON SCALING IN GO

Go is designed for play at several board sizes. However, few machine learning methods can modify the board size in the middle of training and continue learning. This section discusses several exceptions and the following section explains HyperNEAT's approach to scalable Go.

7.3.1 REINFORCEMENT LEARNING AND SCALABLE GO

Because good strategies for 19×19 boards are very different than those for e.g. 9×9, players transitioning from small to large boards must continue to learn and refine their strategy and tactics [Sho03]. Ideally, machine learning algorithms should also learn to play Go at varying board sizes without discarding tactics learned on smaller boards and starting from scratch.

Reinforcement learning has been applied to scalable Go through several approaches [SS08, SSM07, SM04b]. Silver et al. [SSM07] introduced the concept of templates and shape sets. A *template* is defined as a rectangular pattern of pieces smaller than the board that may exist at one or more locations on a Go board. A *shape set* is the set of all possible templates of a given size. During the training process, a weight is assigned to each shape in the shape set. The key idea is that all shapes learned on a smaller board are analogous on a larger one. New shapes that exist only at the higher scale are introduced after scaling by initializing them with a weight of 0. Silver et al. [SSM08], Enzenberger [Enz03], and Schraudolph et al. [SDS94] follow a similar approach.

In a different approach, Stanley and Miikkulainen [SM04b] evolved a neural network for playing Go with NEAT. Instead of acting as a value function, however, the neural network controls a roving eye that has a small field of vision. The robot is able to move across the board and place pieces. Because the field of vision for the robot is smaller than the size of the Go board, the robot can learn local concepts independently of location. As a result, the roving eye can learn to play Go at any resolution.

Schaul and Schmidhuber [SS08] introduced another neuroevolution-based action-value approximator for Go. Unlike the roving eye method, this method is based on the Covariance Matrix Adaptation Evolution Strategy (CMA-ES) [HO01]. Instead of controlling a roving eye that is able to move freely around the board, this approach evolves a Multi-Dimensional Recurrent Neural Network (MDRNN) [GFS07, GS09]. The MDRNN performs swipes across the Go board. To perform a swipe, the same neural network is evaluated at every position

of the Go board. For example, in a up-right swipe, the outputs of the network at position (0,1) and (1,0) are inputs to the network at position (1,1). In this way, information is carried across the board through the output values. MDRNNs are inherently scalable because the network is only concerned with relative information.

While these methods have learned effective Go players, each of them relies on integrating a set of small, local views that are processed independently over time or space. The danger is that less holistic heuristics that are significantly simpler become attractive local optima. In general, an interesting question is whether it is possible to scale the Go board to new resolutions while also processing the entire Go board without relying on subsquares. HyperNEAT creates such a capability, as explained next.

7.4 APPROACH: HYPERNEAT IN GO

Because of the large branching factor in Go [BW95], board evaluation functions such as the HyperNEAT approach to checkers discussed in Chapter 4 may not be tractable in practice. In the case of Go, there can be hundreds of actions to evaluate in a single board state. Thus an appealing alternative would be an *action selector* that evaluates the current state and suggests where to move, rather than a board evaluation function that must view many boards in the future to decide on a move. The next section explores this idea in more detail.

7.4.1 EVOLVING AN ACTION SELECTOR

Because HyperNEAT can evolve high-dimensional structure as an indirect encoding, it opens up the possibility to evolve an action selector. This type of ANN contains an output for *each possible action* (figure 7.2b). In this case, an output exists for each square on the Go board. By activating the substrate, HyperNEAT populates each output with a value indicating the desirability of putting a piece in that position on the Go board. Thus no forward search through the game-tree is needed, thereby saving significant computation. Once the substrate has been activated, the output with the highest activation is chosen and the corresponding square on the Go board undergoes a sanity check that prevents the network from making invalid moves in the game. As a result of this new architecture, the output, hidden, and input layers of the Go substrate all contain $n \times n$ nodes, where n denotes the size of the board. Given a board size of 7×7, the substrate thus contains 147 nodes and 4,802 connections. Indirect encoding can produce the smooth patterns of weights necessary to begin evolution with so many connections and still learn effectively.

The goal of this chapter is to evolve a scalable Go player with the HyperNEAT indirect encoding. To demonstrate scaling, the evolved players compete with a fixed policy opponent at a lower resolution, then scale up to a higher resolution and play the same opponent at the new resolution. To rescale the board this chapter employs discrete substrate extrapolation (Section 5.2).

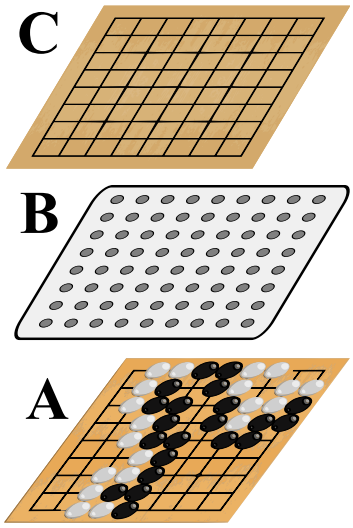


Figure 7.2: **Go Action Selector.** The substrate pictured above is encoded by a CPPN similar to figure 4.1, however this substrate is an action selector instead of a board evaluation function. The substrate contains an output for each possible square on the board. Once the inputs are initialized and the substrate is activated, the outputs contain the desirability of taking the action at the same geometric location as the output node. The outputs corresponding to actions that are not possible because of the rules of the game are ignored.

7.5 EXPERIMENT

The experiment in this chapter aims to determine the effects of scaling HyperNEAT substrates on evolved Go action selectors. The player begins by playing ten games of Go against a fixed policy on a 5×5 board for 500 generations. The fixed-policy player is *Liberty Player* from the SimplePlayers package of Fuego [EM09], who "tries to capture and escape with low liberty stones." A *liberty stone* is surrounded on three of the four sides with stones, and only has one empty adjacent space (i.e. one liberty). Liberty Player can be applied to boards of any size. Because Liberty Player places stones adjacent to stones with few liberties, it escapes captures and also quickly captures given the opportunity. When two or more potential moves are equally viable, Liberty Player picks one at random. These factors make Liberty

Player a nontrivial opponent that provides sufficient challenge to demonstrate the utility of scaling. After training on a 5×5 Go board, the domain switches to playing Go against the same policy on a 7×7 board. Like the evolved player, Liberty Player is an action selector, that is, it only evaluates the current board and returns a location on which to place a stone.

During evolution, each candidate plays ten games of Go against the Liberty Player. After each game has ended, the candidate receives a reward based on the final score and the size of the board:

$$\text{fitness} = \begin{cases} 8b^2 & \text{if the evolved player wins} \\ \max\left(0, s + 2b^2\right) & \text{if the evolved player loses,} \end{cases} \tag{7.1}$$

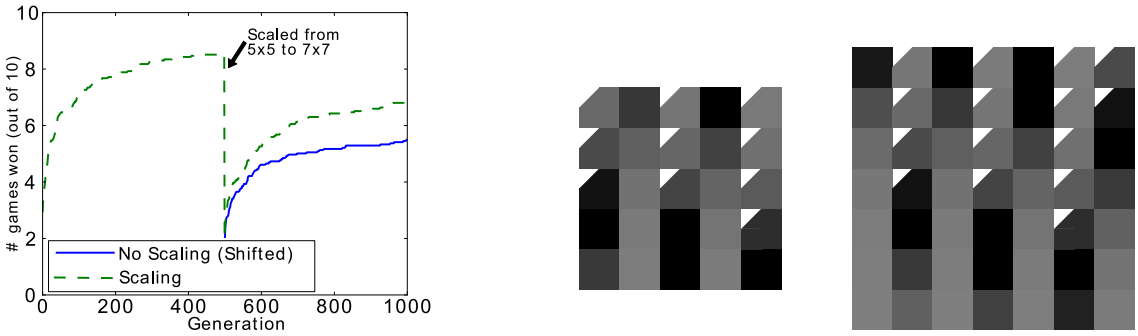
where s denotes the final score and b denotes the size (i.e. length) of the board. This fitness function guarantees that all individuals will receive a positive fitness (as HyperNEAT requires), and that negative Go scores will still result in a positive reward. This convention puts additional emphasis on winning and also avoids rewarding individuals who win by a large margin in a single game, but lose the remaining games. Appendix A details the experimental parameters.

7.6 RESULTS

To determine the effect of scaling, substrate extrapolation is compared to an unscaled approach that plays only 7×7 Go. Although fitness drives evolution, fitness cannot be a benchmark for scaling performance because it is derived from the Go score, which varies with the size of the board. Therefore, the win rate is recorded during evolution and determines the effective skill of the player for the purpose of comparing the scaled to non-scaled methods.

Figure 7.3a compares the performance of the non-scaled 7×7 method against the scaled substrate, averaged over 25 runs. Note that the non-scaled results are shifted to the right so that the reader can easily compare the effects of scaling to not scaling. The scaling approach won significantly more games than the non-scaling approach in all generations after 524 (i.e. 24 generations after scaling) ($p < 0.05$).

To give an idea how scaling works, figure 7.3b shows a single *receptive field* connecting to the center output from the hidden layer of a scalable substrate at the two resolutions. Each grayscale box represents a link weight from a node in the hidden layer at that location to the center node of the output layer. White triangles in the corner of a box denote negative weights. The individual from which this receptive map was extracted is from generation 500, at which the domain is scaled to 7×7. Note that the pattern of weights is extrapolated outward as the substrate is scaled from 5×5 to 7×7. To understand this result, recall that the substrate is scaled with the discrete substrate extrapolation method. As a result, when



(a) Scaled versus non-scaled performance (b) Receptive field at 5×5 and 7×7 scales

Figure 7.3: **Scaling Comparison and Visualization.** The average performance of the generation champions over 25 runs of each variant is shown in (a). The performance is measured as the number of games won out of a possible 10 against Liberty Player. The scaled method wins significantly more than the non-scaled method in every generation beyond 524. A receptive field for the center output node on the substrate is shown in (b). Note that when the substrate is scaled to 7×7, the pattern is extrapolated outwards.

the substrate is created at 5×5, the CPPN is queried with all possible combinations of the numbers $-\frac{2}{3}$,$-\frac{1}{3}$,-0,$\frac{1}{3}$,$\frac{2}{3}$ as inputs x_1, x_2, y_1, y_2. The choice of inputs to the CPPN explicitly defines the particular connection weight that the CPPN will output. The substrate is scaled to 7×7 by expanding the inputs to include all possible combinations of the numbers -1, $-\frac{2}{3}$, $-\frac{1}{3}$, -0, $\frac{1}{3}$, $\frac{2}{3}$, 1. This expansion adds the additional cells shown in 7.3b. This new pattern is thereby an effective bootstrap for learning more advanced concepts at the higher scale.

7.7 IMPLICATIONS

The key contribution of this chapter is to show that indirect encoding makes possible a new kind of holistic, scalable Go player. Interestingly, an evaluation at 7×7 takes *ten times*

longer than the same evaluation at 5×5 because the network size is larger and the games take more turns to complete. A method that can learn fundamental concepts at a low board size can thus more quickly progress to more advanced concepts at higher sizes, and thereby learn them with less computational overhead.

The CPPN encoding allows the HyperNEAT substrate to input and output an entire board of neurons. This method thus differs from other scalable approaches that either divide the board into local segments [SS08] or local features [SSM07]. Constructing a function from the holistic board geometry is important for several reasons. First, it removes the need for a human or external process to divide the search space into local features or segments. Second, constructing functions directly from geometry allow long-distance geometric relationships to be taken into account. For example, the decision to place a piece in Go often hinges not only on the position in the local area, but also on the state of conflicts elsewhere on the board and the geometric relationship of those conflicts with the local positions.

7.8 SUMMARY

This chapter focused on the effects of scaling and demonstrated that players evolved incrementally through a scalable representation learn faster and more effectively than players evolved solely at the large scale. This result implies that fundamental concepts learned at a lower resolution facilitated further learning at the higher scale. The substrate extrapo-

lation method scaled the information learned on the 5×5 Go board to the 7×7 board and the HyperNEAT algorithm was able to continue evolution at this new resolution. The main contribution is a step towards holistic neural strategies through indirect encoding that can be scaled to higher resolution or size. While this method did not search future board positions, search is critical for effective play in strategic and tactical domains. The next chapter reintroduces game-tree search by modifying an action selector substrate to prune search trees.

CHAPTER 8

GEOMETRIC GAME-TREE PRUNING

When humans play board games like checkers or Go, they often do not consider moves that are outside an area of interest. This area of interest depends on the geometric layout of the pieces on the board at the time of evaluation. In traditional game-tree search, every potential move at the current ply is scrutinized. Because HyperNEAT can effectively produce an entire board of outputs, it can potentially evaluate the *entire board* to determine its own area of interest, and then prioritize further game-tree search within those areas (figure 8.1).

Given a prioritization of the board, the alpha-beta search algorithm can then terminate search if no unexplored areas of interest are deemed interesting. Consider that a naive exploration of a game-tree with branching factor b and depth d takes b^d evaluations. Interestingly, assuming a depth of 8, if it were possible to reduce the branching factor from 8 to 7 in a search using geometric pruning, $(8^8)-(7^8) = 11,012,415$ board evaluations could be avoided, a reduction of 34.4% of the original number of evaluations. Thus the idea of *focus*, which plays a strong role in human game playing, can potentially be brought to machine learning. HyperNEAT is a good choice for implementing this idea because its ability to encode high-dimensional networks with many inputs and outputs means that it can evolve a network

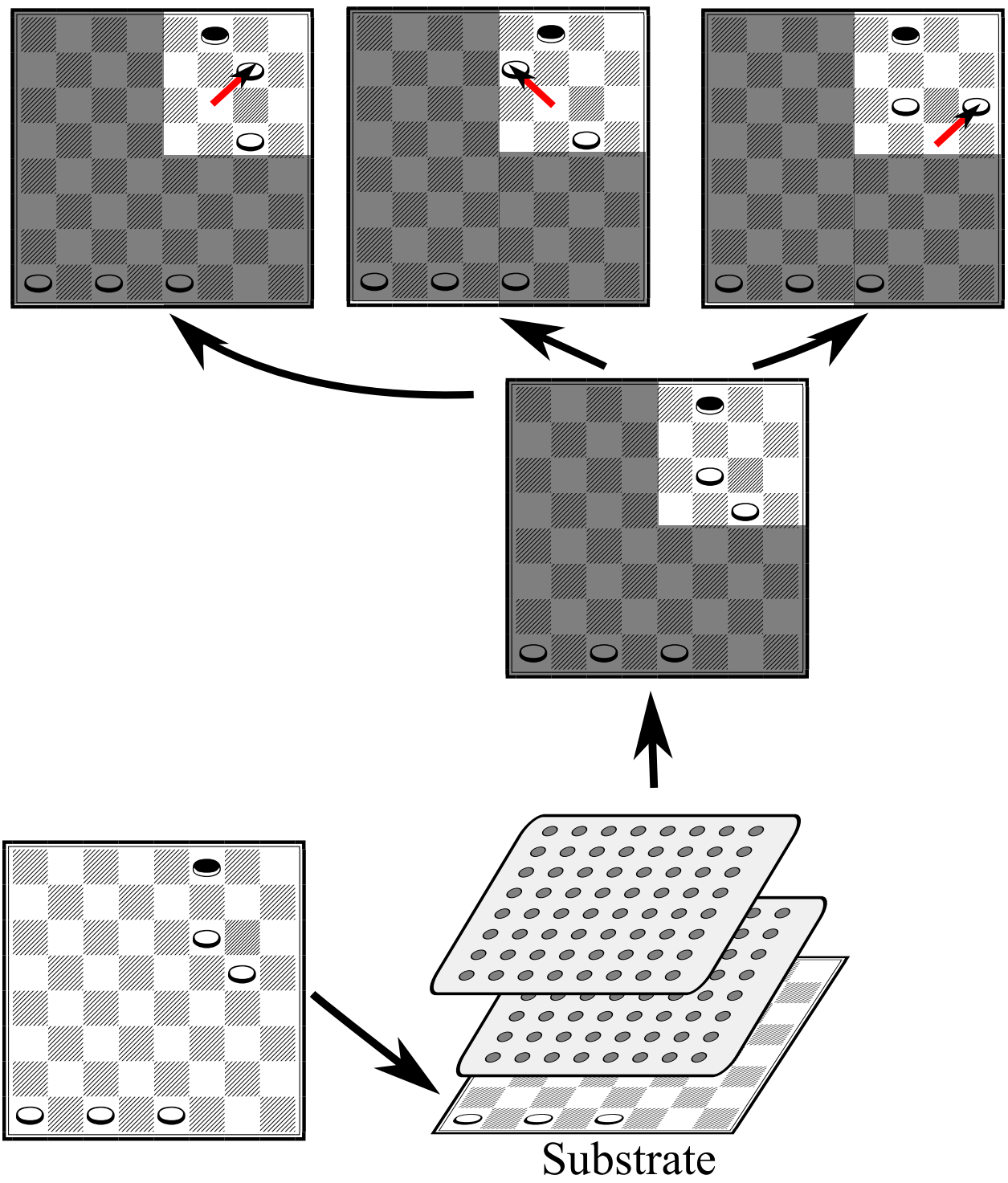


Figure 8.1: **Geometric game-tree Pruning.** The board is presented as an input to a substrate encoded by HyperNEAT. The substrate is activated and the dark squares in the resultant output represent areas of the board that do not deserve focus. In this way, the game-tree search only searches through the paths that involves areas of the board that are given focus by the substrate.

with an output for *every* board position. Each such output can then indicate which moves should be explored, as explained in the next section.

8.1 IMPLEMENTATION

Given a board state of size $n \times n$, HyperNEAT can produce an $n \times n$ output pattern wherein each cell represents the potential *interest* of a move originating from the piece at that position (figure 8.1). This approach is similar to the action evaluation function evolved in Chapter 7. However, instead of explicitly deciding to take the action matching the neuron with the highest activation, geometric game-tree pruning decides what actions *not* to take and employs more traditional game-tree search to decide which action to take among the remaining decisions.

Using geometric game-tree pruning, HyperNEAT is able to focus within its current context and avoid uninteresting moves before they are expanded by alpha-beta search into new game states. This method allows HyperNEAT to search deeper into the game-tree than it otherwise could with the same number of evaluations by pruning moves that are uninteresting.

8.2 EXPERIMENT

The experiment in this chapter focuses on HyperNEAT's ability to play checkers by pruning the game-tree search. In particular, the Cake [Fie08] checkers engine plays moves for both players. Cake is a respected checkers player based on minimax search. Minimax search enumerates all possible actions from the current board state, and then recursively traverses through each of these actions and evaluates the future board states. Once a maximum recursion depth is reached (i.e. the search reaches a position that is enough moves away from the original board state), a heuristic value (computed by Cake) is assigned to the board state and no further actions are chosen from that state. Once the values of all future board states are complete, the set of actions that maximizes the acting player's potential is chosen. HyperNEAT combined with Cake (HyperNEAT-Cake) operates by pruning the enumeration of moves for the current board state, greatly reducing the number of future board states that are evaluated.

While a basic minimax algorithm will search to a given depth d, the Cake engine utilizes iterative deepening [RNC95] to reduce the search space. The intuition behind iterative deepening is to perform a shallow search, and then to prune the search tree of a deeper search with the results from the shallow search. It is important to note that increasing the depth of a search from $d-1$ to d increases the game-tree search space by b^d, where b denotes the branching factor. If we assume that iterative deepening prunes a single move at each evaluation, then iterative deepening reduces the branching factor to $b-1$ at the cost of

performing two searches, one at $d-1$ and one at d. The total cost becomes $b^{d-1}+(b-1)^d$. Lu [Lu93] showed that the average branching factor of checkers is 2.84. Applying this value and the equations of cost for regular search and iterative deepening, iterative deepening would reduce the total number of states visited for all $d > 1$ (i.e. the value 2.84^d is greater than $2.84^{d-1} + 1.84^d$ for all $d > 1$). Cake applies iterative deepening at every possible depth from one to a maximum depth of 99. In addition to reducing the number of states visited, another benefit of iterative deepening to Cake is that it makes it possible to control the total number of states visited. The user can specify a maximum number of states to visit and Cake will iteratively search at increasing depth until the number of states has been reached. If the number of states to visit is kept constant, it is thereby possible for HyperNEAT-Cake to reach a greater depth than regular Cake because HyperNEAT-Cake skips areas of the search tree that HyperNEAT determines to be uninteresting.

The key question of whether HyperNEAT ultimately improves Cake is answered by comparing the performance of trained HyperNEAT-Cake versus regular Cake to the performance of Cake versus itself.

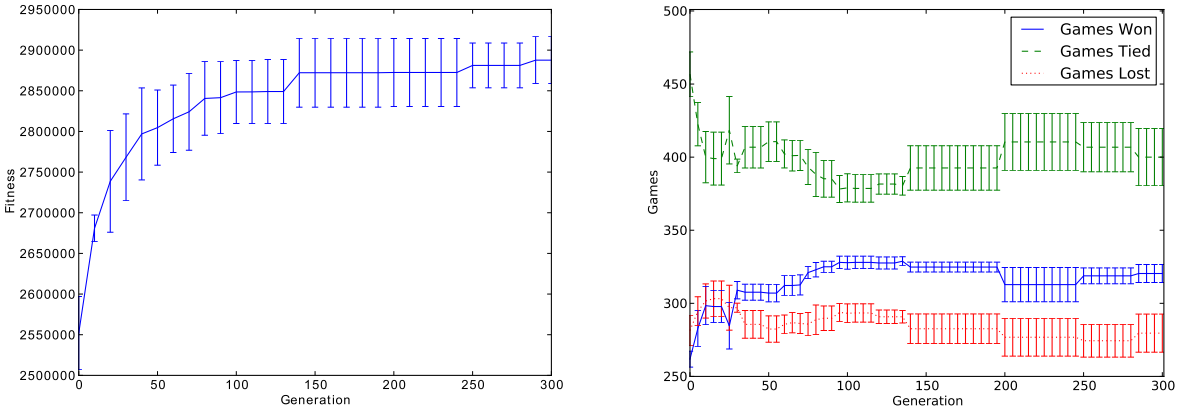
8.3 RESULTS

To evaluate a single individual in the population during training, 25 games are played between HyperNEAT-Cake and regular Cake. Although the heuristic and the search algorithm in

Cake are deterministic, Cake contains many opening book moves that vary among the 25 games. This variation ensures that HyperNEAT-Cake can evolve a sufficiently robust player. The fitness function is the same as in Chapter 4 (Equation 4.1), i.e. fitness is awarded for having material at each round of a game, limiting the opponent's material at each round, and winning a game. Both Cake and HyperNEAT-Cake are limited to 600 search states. When the branching factor for a particular state is three or greater, HyperNEAT-Cake will evaluate the board and then remove the branch corresponding to the action with the lowest output value on the substrate. As a result, HyperNEAT-Cake will not search with as much breadth but is also able to search to greater depth. The question is whether this focus provides an advantage.

To test for generality, 1,000 additional games are played by the generation champion of each generation against the same Cake player from training without the pruning function but with a different and larger set of random seeds. These numbers are compared to the performance of regular Cake against itself to evaluate HyperNEAT's effect on Cake.

Figure 8.2 shows the average performance of HyperNEAT-Cake in training and generalization tests from ten runs. Note that when Cake plays against itself in 1,000 games, on average it scores 213 (stdev=4.9) wins, 229 (stdev=4.7) losses, and 558 (stdev=9.2) ties. In contrast, when the average final generation champion of HyperNEAT-Cake plays against Cake, it wins 320 times (stdev=12.3), loses 279 times (stdev=13.0), and ties 400 times (stdev=39.1). For all ten such champions, the number of wins over their 1,000 games is higher than when Cake plays against itself ($p < 0.05$ by Student's t-test). This result shows that HyperNEAT-Cake



(a) Training Performance (b) Generalization Performance

Figure 8.2: **Training and Generalization Performance of HyperNEAT-Cake.** The average training (a) and generalization (b) performance of the generation champions over ten runs is shown. Adding the HyperNEAT pruning meta-search to Cake resulted in more effective play by learning what to ignore in the traditional alpha-beta search. This result validates the ability of an indirect encoding to learn to act as an adviser to another search algorithm. Note that when Cake plays against itself in 1,000 games, it scores 213 wins, 229 losses, and 558 ties.

is able to prune branches effectively and win significantly more games than regular Cake playing against itself. Thus HyperNEAT-Cake improves upon Cake by learning to decide which parts of the board are not worth exploring further, which is reminiscent of how humans prune their own searches.

8.4 IMPLICATIONS

Search algorithms designed to search the game tree typically are executed online, while the game is being played. In contrast, evolution in this chapter is a kind of meta-search that

attempts to find the best way to constrain the online search. This meta-level knowledge, gained over many games, ultimately yields an understanding of the game that goes beyond the original online search algorithm. As a result, in the case of HyperNEAT-Cake, the search algorithm becomes more focused over the course of many games as the player evolves. The hope is that this added understanding is similar to how humans prune their own searches to only what seems interesting based on past experiences.

Note that the output of HyperNEAT could have a more sophisticated interpretation. At present, the possible moves are enumerated and the move aligned geometrically with the lowest output node on the substrate is removed from the search. Alternatively, the outputs could prioritize the set of available moves and the search algorithm could follow these priorities in a meaningful way to bias the search space. An example of this idea is described in the next chapter, where HyperNEAT is combined with a UCT-based search algorithm.

CHAPTER 9

KILLER APPLICATION: GO

Go (first introduced in Chapter 7) remains a challenge for AI. While in Chapter 7 Go was a test platform for investigating scaling, in this chapter the aim is to investigate the potential for indirect encoding to enhance the state of the art in Go. Current AI approaches to Go have difficulty competing with the most skilled players. Go is challenging mainly because its branching factor is high. Because there are often dozens of potential moves for a given board state, Go requires sophisticated spatial processing and pattern recognition. Because HyperNEAT can learn from geometry, it has the potential to augment the complex spatial processing needed to excel in Go.

9.1 UCT SEARCH

Because Go has a high branching factor, traditional alpha-beta search is not sufficient to effectively explore the game-tree. However, *Upper Confidence bounds applied to Trees (UCT)* search has proven successful in Go [GS07c]. UCT search is a *rollout-based*, *Monte-Carlo* planning algorithm [KS06]. Instead of building a game-tree, rollout-based planning runs

several simulations from the current board state and creates a set of state-action-reward triplets as it completes each simulation. Monte-Carlo planning is a search technique that samples a subset of the moves for a particular board state instead of trying every possible move. Each simulation is run until it reaches a terminating state (i.e. a player has won). As the simulation is running, a list of the state-action pairs chosen over the course of the game is maintained. When the simulation is complete, the state-action pairs for the winning player are rewarded, and the actions taken by the losing player are not rewarded. That is, for a given board state and action (i.e. moving to a square or passing), rollout-based planning stores an expected reward for that state and action.

Once rollout-based planning finishes running simulations, it finally performs a greedy game-tree search, using the state-action-reward triplets as the heuristic. The advantage of rollout-based planning to searching the game-tree recursively is that rollout-based planning automatically takes advantage of the case in which the same state might appear at several different positions in the game-tree. The UCT algorithm combines game-tree search with rollout-based planning and one important addition: As UCT simulates games, it biases the search with the current state-action-reward triplets and current game state information. For example, a UCT implementation may, for a particular state, select the state-action-reward triplet that has the highest reward 50% of the time, but select a random action the other 50% of the time. This bias based on partial solutions encourages exploration of regions in the search space that involve highly-rewarded actions and ignores regions of the search space characterized by less rewarded actions.

Although UCT is a powerful search technique that has proven successful at general game playing [GS08b], vanilla UCT does not explicitly exploit the board geometry in its decision process. Thus there is a promising opportunity for the spatial processing capabilities of HyperNEAT to complement UCT by pruning moves that are less promising. This combination may increase the fidelity of the results produced by the Monte-Carlo simulation.

9.2 GEOMETRY IN UCT FOR GO

Although Go has only one type of move (i.e. putting a token on the board), any unoccupied square on the board is a potential move. To mitigate the myriad of possible moves, expert Go players typically perform two computations. First, they compartmentalize the board into several smaller boards. Second, they memorize responses to certain patterns that may appear anywhere on the board. Humans are able to perceive patterns that involve complex relationships of stones and empty space [M01]. This perception requires a geometric understanding of the Go board, including adjacency, rotational symmetry, and translation.

Section 2.3 showed that CPPNs are effective at representing repetition and symmetry. Thus it follows that CPPNs can potentially effectively represent the patterns necessary to distinguish among various Go scenarios under different geometric transformations.

While there are many potential moves in Go, there are cases in which only specific areas of the board deserve focus. If areas of interest can be described geometrically, HyperNEAT can

capture this geometry and act as an adviser to UCT. Areas of the board that are not marked as interesting by HyperNEAT can be avoided by UCT, thereby significantly speeding it up and allowing for deeper, more informative exploration of the search tree. The next section explains how this idea is implemented in practice.

9.3 IMPLEMENTATION

The goal of this chapter is to demonstrate HyperNEAT's capability to advance the state of the art in Computer Go players. HyperNEAT can enable such an advance by providing an initial bias to the UCT search algorithm. A naive UCT implementation begins with no knowledge of the current board state and must simulate every possible move several times to build a confidence model for each action. As these models become more refined through simulation, UCT biases the search space, investing more search time in actions that have high win rates and disregarding actions with low win rates. While a naive UCT implementation may be effective in domains with a low branching factor, such an approach is not sufficient for Go because the branching factor is prohibitively high. To address this problem, many UCT Go engines rely on human engineering to estimate the win-rate for new actions without visiting them. As a result, UCT begins with a heuristic-driven confidence model for each action. Such an estimation is often ad hoc and difficult for humans to formalize. HyperNEAT

creates the opportunity to replace the human engineer by evolving initial models for unseen actions that UCT can then refine through simulation.

To investigate HyperNEAT's ability to enhance the performance of UCT, a proven Go engine is a good place to start. For this purpose, the UCT Go implementation from the Fuego Go engine [EM09, Mul09] is chosen to combine with HyperNEAT. Fuego was a top-tier Computer Go player in the 2009 Computer Olympiad Go Tournament, winning the 9×9 Go competition and placing second in the 19×19 competition. It is important to note that in the experiment in this chapter, *only* the UCT engine of Fuego is in play, and the pattern matching and default knowledge modules of Fuego are omitted. That way, the ability of HyperNEAT to enhance UCT is isolated. By default, the UCT algorithm assigns a 40% win-rate to all possible actions as a bootstrap mechanism before any actions are chosen. HyperNEAT is integrated with UCT by replacing this value with outputs from the substrate. Instead of assigning the default value of 40%, a HyperNEAT substrate is activated that contains an output for each square on the board. The output with the same geometric location as the action is queried to represent the initial win-rate for that action. Because the activation of a node is in the range $(-1.0, 1.0)$, the activation is linearly normalized to fit the range $(0.0, 1.0)$. This initial win-rate is updated later by the UCT algorithm based on the results of its simulation. The key insight is that UCT will bias its search based on the win-rates output by HyperNEAT. As a result, HyperNEAT is driving the focus of UCT to regions of the board that appear interesting.

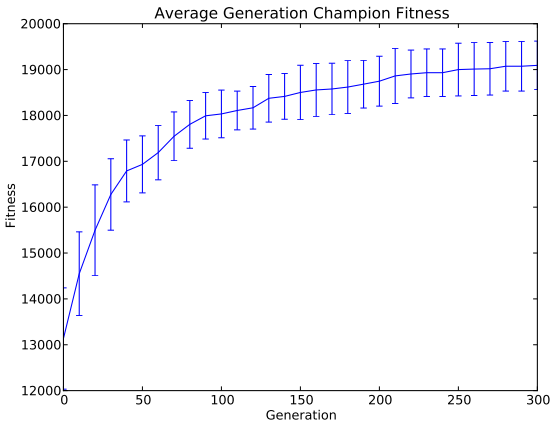
9.4 RESULTS

During training, the HyperNEAT-UCT hybrid (HyperNEAT-UCT) plays 20 games against regular UCT. Each game is unique because a unique random seed is assigned to the UCT algorithm, which affects the opening book moves chosen and the simulations. The number of UCT simulations per move is set to 1,000. Fitness is awarded based on the final score from Equation 7.1 from Section 7.5. To test for generality, 1,000 games are played with unique random seeds by the candidate player. Both 5×5 Go and 9×9 Go runs are attempted. The evolutionary parameters are documented in Appendix A.

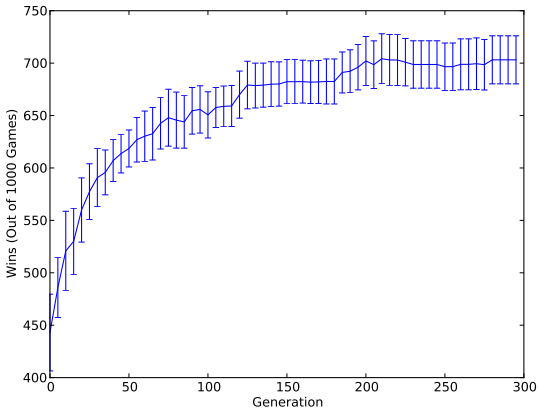
The results from these experiments and the generalization tests for 5×5 and 9×9 are shown in Figure 9.1. By achieving win-rates of 70.9% (stdev=4.0%) in 5×5 Go and 87.3% (stdev=2.0%) in 9×9 Go on average in the last generation, these results show that HyperNEAT-UCT is able to play more effective Go than UCT without prior knowledge, confirming that HyperNEAT's ability to exploit geometry can indeed complement the raw search of UCT.

9.5 IMPLICATIONS

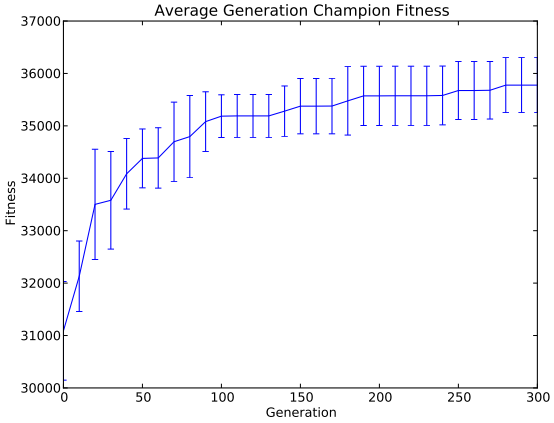
As in Section 8.4, combining HyperNEAT with UCT once again enhances an online search process with knowledge evolved over many games. This technique is especially valuable



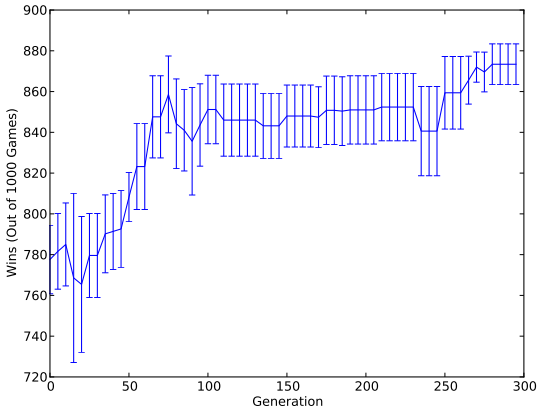
(a) Training Performance at 5×5



(b) Generalization Performance at 5×5



(c) Training Performance at 9×9



(d) Generalization Performance at 9×9

Figure 9.1: **HyperNEAT-UCT Training and Generalization Performance.** The average training and generalization performance of the generation champions over 10 runs for 5×5 (a & b) and 9×9 (c & d) are shown. In 5×5, the most general individual won 788 of 1,000 games, a win-rate of 78.8%, while in 9×9 the most general individual won 894 of 1,000 games, a win-rate of 89.4%.

for games such as Go, in which the patterns on the board often suggest specific regions that should be explored to more depth or avoided completely. Assigning win-rates to every intersection on the board is only possible because HyperNEAT can evolve a network with a large output array.

It is interesting to note that the initial individuals actually perform relatively well in training and generalization (e.g. the win-rate in generalization for generation 1 in 9×9 Go is 77.76% on average). This tendency can be attributed to the positive bias of CPPN encoding. In the first generation, the CPPNs are constructed by randomly assigning weight values to a fully connected CPPN with no hidden nodes. In all runs, at least one of the CPPNs from the first generation is shown to contribute in a positive way to the performance of UCT. This fact is partly due to the pruning mechanism itself providing an advantage in some scenarios, even without an optimal choice of moves to prune; however, the regular nature of CPPNs and the regular nature of checkers also complement each other. Simple CPPNs that are successful in checkers typically subtract the x and y coordinates to tessellate the same rule across the entire board space. This pattern complements the rules of Go, which themselves are tessellated across the game board.

The techniques for bootstrapping the UCT search is what separates a world class computer Go player from a hobbyist UCT Go implementation. Many of these techniques are hand-coded by experts and tweaked repeatedly based on the results from games with other computer players. The results in this chapter suggest that UCT can be bootstrapped automatically by a learning algorithm, in this case a neuroevolution algorithm with an indirect

encoding. The key challenge for the future is to combine this bootstrapping process with preexisting sophisticated heuristics, such as in complete Fuego. If HyperNEAT can improve upon such heuristics, it may impact the state-of-the-art in computer Go. The results in this chapter are an important step in that direction.

CHAPTER 10

DISCUSSION AND FUTURE WORK

HyperNEAT is an indirect encoding that is computed as a function of geometry. As a result, it is aware of the domain geometry and can exploit this knowledge. This chapter begins with a discussion of the importance of learning from geometry, and then explains how learning from geometry can benefit tactical and strategic decision-making domains. The following section then contemplates the two extensions: substrate extrapolation and geometric game-tree pruning. The final section covers future possibilities created by the results in this dissertation.

10.1 LEARNING FROM GEOMETRY

A major motivation for ANNs is that they are inspired by real neural networks found in the brain. Yet traditional ANNs have not reached the complexity of real brains. One factor that contributes to this gap is in how natural brains develop. While most ANN training algorithms begin with random values for each connection weight, the initial connectivity patterns within the brain of an embryo develop many regular patterns [Spo02, BK99, HKB94,

HP02, SM03]. The DNA that encodes a human brain includes 30,000 genes, yet it encodes 100 trillion connections [Del95, KSJ91, DSG98]. Because many organisms exhibit symmetry and repetition both in their bodies and in their brains, it is clear that DNA produces such symmetries as part of its highly compressed encoding.

Current methods with neural networks rely on human engineering to decompose a large input space (e.g. a picture with several thousand pixels) into a smaller space (e.g. a set of features) [LS04]. This task requires domain-specific knowledge, and also adds a human element that involves time-consuming trial and error on the part of the engineer. In contrast, neuroevolution algorithms based on an indirect encoding can unfold a relatively small genotype into a large phenotype that is able to directly interact with large state spaces, thereby eliminating the need for a human engineer to decompose the input space explicitly.

CPPNs are an effective indirect encoding because they are grounded in geometry. By treating Cartesian coordinates as inputs, CPPNs literally create patterns as a function of geometry. The effect is highly compact encoding. For examples, in the boxes domain (Section 5.1), a CPPN with 17 connections is able to encode a substrate with over eight million connections. This level of compression begins to approach the compression exhibited by DNA. Assuming that geometry and locality are critical to the mapping between DNA and the embryo, CPPNs are a step towards reproducing the powerful encoding evident in nature. HyperNEAT is a novel developmentally-motivated encoding able to learn from geometry and thereby discover geometric regularities such as symmetry and repetition.

10.2 GEOMETRY IN TACTICAL AND STRATEGIC DECISION-MAKING DOMAINS

Tactical and strategic decision-making domains often are described by a simple set of rules yet require complex strategies. These strategies are tied to the spatial arrangement of the board, and how such arrangements result in unique and interesting applications of the rules (e.g. moving a piece in checkers is often a simple diagonal shift, but may require jumps, multi-jumps, and crowning based on the board arrangement). Awareness of geometry is potentially important across all methods in machine learning. Simply imagining trying to learn tic tac toe, a game that is usually simple to understand, on a scrambled board illustrates this insight.

The experiments with checkers (Chapters 4 and 8) show that geometry plays a critical role in tactical and strategic decision-making domains. NEAT-EI (the version with engineered inputs) is able to defeat the SimpleCh heuristic because it has engineered inputs based on domain expertise. In contrast, NEAT by itself was not able to defeat the opponent at all, even in 600 generations. Yet the problem with engineering inputs is that encoding regularities by hand involves a labor-intensive process and domain expertise, both of which may be expensive or impossible depending on the domain. Learning from geometry is an attractive alternative to engineering inputs, and in the case of the checkers experiment, HyperNEAT was able to outperform NEAT-EI by defeating the heuristic within 20 generations in every run. HyperNEAT learns quickly because it constructs substrates as functions of

geometry. The CPPN that encodes the solutions to the experiment are in effect able to see the adjacencies and symmetries present in the domain.

10.3 HYPERNEAT EXTENSIONS

Three extensions to the HyperNEAT method are introduced in this dissertation. The first is the *action-selector* substrate architecture, which makes it possible to choose a move without any game-tree search. Networks large enough to support action-selection only become possible through indirect encoding because they require an output for every possible action.

The second is *substrate extrapolation*, whereby the CPPN encodes a small substrate through the first phase of evolution, and then increases the resolution of the substrate as evolution progresses. This form of incremental evolution can speed up the evolutionary process. Substrate extrapolation is possible because CPPNs are able to encode substrates of any size. Two forms of substrate extrapolation were introduced and an experiment for each form of substrate extrapolation was presented.

To demonstrate continuous substrate extrapolation, a visual discrimination task (i.e. the boxes domain) was selected (Section 5.1). While intuitively simple, the boxes domain revealed a wide disparity in the ability of directly and indirectly encoded ANNs to solve problems with high dimensionality. HyperNEAT with continuous substrate extrapolation was able to evolve a solution to the boxes domain that scaled with the domain and included

networks with over eight million connections. Networks of this magnitude had never before been evolved from scratch with a neuroevolution algorithm, but problems with high dimensionality are now tractable through indirect encoding.

Discrete substrate extrapolation was demonstrated through the game of Go against a fixed-policy opponent. HyperNEAT was able to evolve a player at the 5×5 Go board size, and then scale to 7×7 and continue evolution. This form of bootstrapping was shown to outperform simply learning the 7×7 game directly. These results suggest that discrete substrate extrapolation allows indirectly encoded learning algorithms to learn faster and ultimately learn better by scaling the domain during evolution than by learning the higher dimensional problem directly.

The third extension to HyperNEAT is *geometric game-tree pruning*. This idea introduces the concept of *focus* to neuroevolution, which allows potential actions in the search algorithm to be pruned based on geometry. Because CPPNs are able to effectively encode large substrates without a reduction in performance, it is possible to create a substrate that contains many outputs (i.e. one for each square on the board). These outputs can then in concert prune out areas of the board that are uninteresting. In this way, geometric game-tree pruning is a meta-search that learns to optimize the local search algorithm offline through the course of many games. HyperNEAT thus captures the global properties of the domain, and applies this knowledge to bias the local search algorithm. This approach was shown to create more effective Checkers players.

Finally, the dissertation proposed Go as a killer application. Go is well suited to benchmarking geometric processing and currently receives a lot of attention from the artificial intelligence community [GS07b, Mul09, M01, Enz03]. The proposed extensions can potentially improve the performance of AI in Go beyond the current state of the art. In fact, they ultimately produce a new UCT-HyperNEAT hybrid algorithm. This algorithm proved significantly more effective than UCT without any bias.

10.4 FUTURE WORK

The results in this dissertation demonstrate an innovative approach to Computer Go, but do not include playing Go on a regulation-sized (i.e. 19×19) Go board; nor do the evolved networks play at the level of the top players (e.g. Fuego, MoGo, Many Faces of Go). Future work should include expanding on the results in this dissertation to include larger board sizes. Furthermore, while HyperNEAT was able to improve upon UCT, an interesting prospect for future work is to invoke the HyperNEAT-trained ANN at appropriate times in a system like the complete Fuego [Mul09, EM09], which has been ranked the best computer player at 9×9 Go. The trick would be to decide when to follow Fuego's heuristics and when to follow instead the recommendation of the substrate during UCT search. The hope is that the combination could produce a new world-class player.

Board games like Go and checkers are interesting because their tactics and strategies are also relevant to problems in the real world. Interestingly, any domain that exists in a space of multiple dimensions contains at least an implicit geometry that can be potentially exploited through an indirect encoding based on geometry. Clune et al. [COP08a] demonstrate that, even when the geometry of an ANN that controls a robot is scrambled, HyperNEAT is able to find regularities within the scrambled geometry. Thus future work for this approach will also explore other challenging domains. Robot vision is clearly geometric [GS07a] because the field of vision is captured by the eye on a sheet of photoreceptors; however robot control [DS07] can also be interpreted geometrically, as is done in the hippocampus region of the brain. While early such work focused on relatively simple problems, it is not known how close evolved indirect encodings can approach the complexity of biological brains, which are clearly suited for such tasks. Even if approaches such as HyperNEAT do not reach such ambitious scale, lessons learned along the way, such as the connection between smooth geometry and generalization (Chapter 5.2), promise to be illuminating.

For example, an interesting question is whether ANNs evolved by HyperNEAT for visual tasks might resemble features in V1 or other parts of the biological visual processing hierarchy [BM06, Hub88]. While the human primary visual cortex contains about 140 million neurons [LK94], HyperNEAT has evolved functional networks with millions of connections [SDG09]. Furthermore, while biological brains (including the visual cortex) exhibit synaptic plasticity [HWL77], ANNs with plastic synapses have been evolved in the past [RVH09, SBM08b, FU00, FU01, FM96], and in principle HyperNEAT can potentially evolve the *geometry of*

the learning rules [RS10], taking it another step closer to biological plausibility. That is, HyperNEAT can potentially assign plasticity roles to connections in a geometric pattern, which is necessary if plastic structures with millions of connections are to be evolved. Thus, while the evolved maps in this dissertation are static, in principle the capability to encode such maps suggests that evolving plastic maps with similar properties (e.g. geometry) is plausible.

CHAPTER 11

CONCLUSION

This dissertation introduced the Hypercube-based Neuroevolution of Augmenting Topologies (HyperNEAT) method for evolving Compositional Pattern Producing Networks (CPPNs) that encode neural network as a pattern of network weights. The approach was tested and enhanced in tactical and strategic decision-making domains. This chapter summarizes the main features of the dissertation.

11.1 CONTRIBUTIONS

Five major contributions were presented that together move the field of neuroevolution significantly forward:

1. HyperNEAT was introduced. It contains a unique indirect encoding of ANNs that has proven effective in several domains [GS08a, GS07a, GS10a, GS10b, DLR10, DS07, DS08, RS10, CBO09b, CBP09, CBO09a, CPO09, CBM10, COP08b, VS10b, VS10a, SDG09]. As machine learning methods tackle more difficult problems in tactical and strategic decision-making domains, representation is becoming increasingly critical to

effective learning. The process of converting state information into ANN inputs is challenging for human engineers; however, biology has shown that elegant mappings from the real world are possible. With CPPNs that see geometry and encode ANNs based upon it, it is possible for evolution to discover complex geometric regularities as *functions of geometry* rather than through human intuition.

2. A novel ANN topology was introduced that evaluates all actions for a given decision simultaneously. The action selector is more effective in tactical and strategic domains than the traditional state evaluation topology because the action selector topology both saves search time and also integrates better with search algorithms such as minimax and UCT.

3. Substrate extrapolation enables the encoding to represent several ANNs, each at a unique scale but each containing the same general motifs in its connectivity. Two variants of substrate extrapolation were introduced: continuous substrate extrapolation and discrete substrate extrapolation. These variants enable scaling the resolution and size of the neural network topology. Continuous substrate extrapolation was demonstrated in a robot vision domain and scaled to networks with millions of connections while continuing to solve the vision task even as the resolution of the input was allowed to vary. Representing a vision problem with an input neuron for each pixel in the image through an encoding that is invariant in performance with the resolution of the image was previously not tractable in neuroevolution. Discrete substrate extrapolation allowed HyperNEAT to train against the LibertyPlayer Go engine in 5×5 Go, then

scale to 7×7 Go and continue evolution. The results showed that the information learned on the smaller scale allowed rapid progress during the continued evolution at the larger scale.

4. Geometric game-tree pruning was shown to give focus to game-tree search based on the spatial characteristics of the domain. This focus allowed the Cake checkers engine to explore to a greater depth than normal without increasing the size of the search space by eliminating moves that were outside of the area of interest.

5. Finally, HyperNEAT-UCT was constructed, wherein HyperNEAT evolved action win-rate probabilities for the UCT algorithm that then explored actions with a bias towards more winning actions in Go. The result was an improvement over UCT alone, which is a core element of top-tier players like Fuego.

The results and analysis in this dissertation ultimately suggest that an important prerequisite to exploiting geometry in learning is to be *aware* of it. That the CPPNs in HyperNEAT literally see the positions of the nodes being connected affords the ability to exploit the domain geometry by creating smooth, semi-regular patterns. To date, this ability to *see* the geometry of the substrate is unique, yet it portends the importance of endowing future algorithms with a similar capability if they are to exploit domain geometry effectively. Once the capability to perceive geometry is made available, an exciting new research direction with interesting biological parallels opens up.

11.2 CONCLUSION

HyperNEAT is a new method for encoding neural networks that are a function of the geometry in which they are embedded. The extensions to HyperNEAT, action selection, substrate extrapolation and geometric game pruning, provide HyperNEAT with a new capability for indirect encodings. HyperNEAT with these extensions was able to combine with UCT to create an improved Computer Go player.

APPENDIX: PARAMETER VALUES

This appendix describes system parameters and their values across all the experiments in this dissertation. The appendix is divided into three sections. The first section gives a brief summary of each parameter. The second section describes the values of all parameters that are constant across all experiments. The final section outlines the population sizes and generation counts for each experiment.

1 PARAMETER GLOSSARY

The parameters discussed in this section are those inside the C++ implementation of HyperNEAT by myself in which all experiments in this dissertation were performed. It is available at `http://eplex.cs.ucf.edu/software.html#gaucij_HyperNEAT`.

- The number of individuals in the population is the **population size**. All of these individuals are evaluated at every generation and a new generation of individuals is created from the originals.

- The number of generations in an experiment is called the **generation count**. Once the number of generations has reached the generation count, the final population is evaluated and then the experiment ends.

- The **disjoint node coefficient** (C_d)**, excess node coefficient** (C_e)**,** and **weight difference coefficient** (C_w) parameters calculate how different (i.e. incompatible)

two individuals are for the purpose of speciation. Assuming D disjoint connections, E excess connections, and an average weight difference of $\bar{W}$, the overall compatibility distance between two individuals i_1 and i_2 is computed as follows:

$$\text{Compatibility}(i_1, i_2) = C_d D + C_e E + C_w \bar{W} \tag{11.1}$$

- Individuals must be compared to each other to determine the species. If the difference between two individuals (based on Equation 11.1) is greater than the **compatibility threshold**, the individuals will be placed in different species.
- The **species size target** is the target number of species that HyperNEAT attempts to maintain each generation. Although this number is the target, the actual number of species may be less than or greater than this number for a given generation.
- The compatibility threshold is a dynamic parameter, and changes with each generation based on the species size target parameter and the actual number of species in the population (i.e. the compatibility threshold decreases if the number of species is less than the species size target parameter and vice versa). The **compatibility modifier** determines the magnitude that the compatibility threshold changes each generation.
- The **survival threshold** indicates the percentage of individuals allowed to reproduce from each species.

- At each generation, the maximum fitness of each species is computed. If a species does not improve in fitness after a certain number of generations defined by the **dropoff age** parameter, new individuals in the species will not reproduce.

- Most offspring are created by performing crossover between two parents in the same species; however the **mutate only probability** parameter defines a chance that an offspring will be a complete copy of a single parent.

- After offspring are created from one or two parents of the previous generation, the **mutate add node probability** and **mutate add link probability** parameters determine the chance that a new node or link will be added to the offspring. The offspring also has a chance to have its link weights mutated, defined by the **mutate link weights** parameter. If the offspring is chosen for link weight mutation, the **mutate link probability** defines the chance that a particular link will be mutated. This chance is applied to every link in the genome.

- By default the activation function for each node (in the genome, not the substrate) is sigmoid. The default can be changed to Gaussian by setting the **only Gaussian hidden nodes** parameter. When new nodes are added or nodes are mutated, it is possible for their activation function to change if the **extra activation functions** parameter is set. This option should also be chosen if the user is evolving CPPNs.

- When a new node is inserted into a CPPN or directly-encoded ANN by mutation, it is inserted between two nodes that are already connected by a link, and two additional

links are added connecting the new node to both existing nodes. If the **add bias to hidden node** parameter is set, a third link is added to the new node from the bias node. Note that even if this parameter is not set, a link might still be added between new nodes and the bias node in future mutations.

- Each node and link has an age, which is defined as the difference between the current generation and the generation that the new node or link was created. If a link age is less than the **adult link age** parameter, then the mutate link probability is ignored and the link is mutated 100% of the time.

- The **mutation power** parameter defines the maximum possible change in a link weight from a single mutation. The actual change for a single mutation is a random number in the range $(-x, x)$, where x is the mutation power.

- By default, new nodes are not added to recurrent connections; however that can be changed by setting the **allow add node to recurrent connection** parameter.

- Elitism for the population is in effect if the **force copy generation champion** parameter is on; however elitism for each species still depends on the species size. If a species size is smaller than the **smallest species size with elitism** parameter, the species champion will not be copied to the next generation.

- CPPNs can be allowed to have recurrent connections, and as a result it is not intuitive to determine how many times to activate the network before evaluating the output

nodes. The default number of times is $1+u$ where u is the **extra activation updates** parameter.

- All of the networks evolved in this dissertation contain nodes with signed activation, but it is possible to create networks with unsigned activation by setting the **signed activation** parameter to zero.

- The **random seed** parameter specifies a seed for the random number generator that the internal NEAT and HyperNEAT evolution algorithms draw numbers from. Note that this generator is independent from any other random number generators in any of the experiments. The user can specify a -1.0 and HyperNEAT will generate its own seed based on the current time in seconds and the number of clock ticks since the start of execution. Note that HyperNEAT will fill the actual seed generated into the XML output, and not the value -1.0. In this way, it is possible to rerun the same experiment with the random seed in the XML output and verify that the algorithm is deterministic.

- The **experiment type** parameter determines which experiment to execute. If a new experiment is added, the experiment type must be defined in the HyperNEAT source.

- To save disk space, the XML output does not contain every individual in all generations of the population. Typically, the generation champion for each generation and every individual in the final generation are saved to disk. It is possible to save all of the individuals in intermediate generations. All of the individuals for every x generations

are saved to disk, where x is the **generation dump modulo** parameter. Setting this parameter to zero enforces the default behavior mentioned above.

2 FIXED PARAMETERS

The parameters in this section were the same in all experiment in this dissertation. Because HyperNEAT is based on NEAT, the parameters largely reflect traditional parametrizations in regular NEAT. The disjoint and excess node coefficients were both 2.0 and the weight difference coefficient was 1.0. The compatibility threshold was 6.0 and the compatibility modifier was 0.3. The survival threshold was 20%. The target number of species was eight and the drop off age was 15. The survival threshold within a species was 20%. Offspring were created by two parents 75% of the times, and created from asexual reproduction the remaining time. Offspring had a 3% chance of adding a node, a 5% chance of adding a link, and an 80% chance of link mutation. If chosen for link mutation, each link of the new offspring had a 10% chance of being mutated. Available CPPN activation functions were sigmoid, Gaussian, sine, and linear functions. Recurrent connections and self-recurrent connections within the CPPN were not enabled. Signed activation was used, resulting in a node output range of $[-1, 1]$. So that activation could travel through all the structure in an arbitrary CPPN topology, each CPPN was activated 10 times for each query before its output was assigned as the connection weight in the substrate. By convention, a connection

Experiment	Population Size	Generation Count	Target Species Num.
Checkers	120	200	15
Visual Discrimination	100	300	16
Scalable Go	100	500	16
Checkers game-tree Pruning	100	300	15
HyperNEAT-UCT Go 5×5	100	300	15
HyperNEAT-UCT Go 9×9	50	300	15

Table 11.1: **Population Sizes, Generation Counts, and Target Number of Species by Experiment.** The population size, generation count, and target number of species are shown for each of the experiments described in this dissertation.

is not expressed if the magnitude of its weight is below a minimal threshold of 0.2 [GS07a]; otherwise, it is scaled proportionally to the CPPN output. The number of extra activations is set to nine, and the adult link age was set to 18. These parameters were found to be robust to minor variation in preliminary experimentation.

3 POPULATION SIZE AND GENERATION COUNT

The population size, species number, and generation count vary from experiment to experiment. This variation exists because (1) some evaluations are so expensive that a high population size would take a prohibitively long time and (2) some domains require more time before the fitness plateaus. Table 11.1 shows the population size and generation count for each experiment.

Bibliography

[Alt94] Lee Altenberg. "Evolving Better Representations through Selective Genome Growth." In *Proceedings of the IEEE World Congress on Computational Intelligence*, pp. 182–187, Piscataway, NJ, 1994. IEEE Press.

[Ang95] P. J. Angeline. "Morphogenic Evolutionary Computations: Introduction, Issues and Examples." In John Robert McDonnell, Robert G. Reynolds, and David B. Fogel, editors, *Evolutionary Programming IV: The Fourth Annual Conference on Evolutionary Programming*, pp. 387–401. MIT Press, 1995.

[ASP93] Peter J. Angeline, Gregory M. Saunders, and Jordan B. Pollack. "An Evolutionary Algorithm That Constructs Recurrent Neural Networks." **5**:54–65, 1993.

[kstanley: Evolutionary algorithm with only mutation for TWGANN.]

[BA83] P. Burt and E. Adelson. "The Laplacian pyramid as a compact image code." *IEEE Transactions on communications*, **31**(4):532–540, 1983.

[BG92] R.D. Beer and J.C. Gallagher. "Evolving dynamical neural networks for adaptive behavior." *Adaptive behavior*, **1**(1):91, 1992.

[BK99] Petet J. Bentley and S. Kumar. "Three Ways to Grow Designs: A Comparison of Embryogenies for an Evolutionary Design Problem." In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-1999)*, pp. 35–43, 1999.

[BKM02] James A. Bednar, Amol Kelkar, and Risto Miikkulainen. "Modeling Large Cortical Networks with Growing Self-Organizing Maps." *Neurocomputing*, **44–46**:315–321, 2002. Special issue containing the proceedings of the CNS*01 conference. *[Combined citation for clarity when submitting bednar:neuroinformatics04.]*

[BM06] James A. Bednar and Risto Miikkulainen. "Joint Maps for Orientation, Eye, and Direction Preference in a Self-Organizing Model of V1." *Neurocomputing*, **69**(10–12):1272–1276, 2006.

[Bon02] Josh C. Bongard. "Evolving Modular Genetic Regulatory Networks." In *Proceedings of the 2002 Congress on Evolutionary Computation*, 2002.

[Bot08] Jack Botermans. *The Book of Games: Strategy, Tactics, and History.* Sterling Publishing Co., 2008.

[BW93] Heinrich Braun and Joachim Weisbrod. "Evolving Feedforward Neural Networks." 1993.

[kstanley: ENZO- uses connection-specific distance coefficients to allow crossover in TWGANN.]

[BW95] J. Burmeister and J. Wiles. "The challenge of Go as a domain for AI research: A comparison between go and chess." In *Proceedings of the Third Australian and New Zealand Conference on Intelligent Information Systems. IEEE Western Australia Section*, pp. 181–186, 1995.

[CBM10] J. Clune, B.E. Beckmann, P.K. McKinley, and C. Ofria. "Investigating whether HyperNEAT produces modular neural networks." In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2010)*, New York, NY, 2010. ACM Press.

[CBO09a] J. Clune, B.E. Beckmann, C. Ofria, and R.T. Pennock. "Evolving Coordinated Quadruped Gaits with the HyperNEAT Generative Encoding." *IEEE Congress on Evolutionary Computation*, 2009.

[CBO09b] Jeff Clune, Benjamin E. Beckmann, Charles Ofria, and Robert T. Pennock. "Evolving Coordinated Quadruped Gaits with the HyperNEAT Generative Encoding." In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC-2009) Special Session on Evolutionary Robotics*, Piscataway, NJ, USA, 2009. IEEE Press.

[CBP09] J. Clune, B.E. B.E. Beckmann, R.T. Pennock, and C. Ofria. "HybrID: A Hybridization of Indirect and Direct Encodings for Evolutionary Computation." In *Proceedings of the European Conference on Artificial Life (ECAL-2009),*, 2009.

[CF99] Kumar Chellapilla and David B. Fogel. "Evolution, Neural Networks, Games, and Intelligence." *Proceedings of the IEEE*, **87**:1471–1496, 1999.

[CF01] Kumar Chellapilla and David B. Fogel. "Evolving an expert checkers playing program without using human expertise." *IEEE Trans. Evolutionary Computation*, **5**(4):422–428, 2001.

[Chu86] Paul M. Churchland. "Some reductive strategies in cognitive neurobiology." *Mind*, **95**:279–309, 1986.

[CK04] Dmitri B. Chklovskii and Alexei A. Koulakov. "MAPS IN THE BRAIN: What Can We Learn From Them?" *Annual Review of Neuroscience*, **27**:369–392, 2004.

[COP08a] J. Clune, C. Ofria, and R. T. Pennock. "How a Generative Encoding Fares as Problem-Regularity Decreases." In *Proceedings of the 10th international conference on Parallel Problem Solving from Nature*, pp. 358–367, Berlin, Heidelberg, 2008. Springer-Verlag.

[COP08b] J. Clune, C. Ofria, and R.T. Pennock. "How a Generative Encoding Fares as Problem-regularity Decreases." In *Proceedings of the 10th International Conference on Parallel Problem Solving From Nature (PPSN 2008)*, pp. 258–367, Berlin, 2008. Springer.

[COP09] J. Clune, C. Ofria, and R.T. Pennock. "The sensitivity of HyperNEAT to different geometric representations of a problem." In *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, pp. 675–682. ACM, 2009.

[CPO09] Jeff Clune, Robert T. Pennock, and Charles Ofria. "The Sensitivity of HyperNEAT to Different Geometric Representations of a Problem." In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2009)*, New York, NY, USA, 2009. ACM Press.

[Del95] Frank Dellaert. *"Toward a Biologically Defensible Model of Development."*. Master's thesis, Case Western Reserve University, Clevekand, OH, 1995.

[DLR10] David D'Ambrosio, Joel Lehman, Sebastian Risi, and Kenneth O. Stanley. "Evolving Policy Geometry for Scalable Multiagent Learning." In *Proceedings of the Ninth International Conference on Autonomous Agents and Multiagent Systems (AAMAS-2010)*, pp. 731–738. International Foundation for Autonomous Agents and Multiagent System, 2010.

[DS07] David D'Ambrosio and Kenneth O. Stanley. "A Novel Generative Encoding for Exploiting Neural Network Sensor and Output Geometry." In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2007)*, New York, NY, 2007. ACM Press.

[DS08] David B. D'Ambrosio and Kenneth O. Stanley. "Generative Encoding for Multiagent Learning." In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2008)*, New York, NY, 2008. ACM Press.

[DSG98] P. Deloukas, G. D. Schuler, G. Gyapay, E. M. Beasley, C. Soderlund, P. Rodriguez-Tome, L. Hui, T. C. Matise, K. B. McKusick, J. S. Beckmann, S. Bentolila, M. Bihoreau, B. B. Birren, J. Browne, A. Butler, A. B. Castle, N. Chiannilkulchai, C. Clee, P. J. Day, A. Dehejia, T. Dibling, N. Drouot, S. Duprat, C. Fizames, and D. R. Bentley. "A Physical Map of 30,000 Human Genes." *Science*, **282**(5389):744–746, October 23 1998.

[EM09] M. Enzenberger and M. Müller. "Fuego–an open-source framework for board games and go engine based on monte-carlo tree search." Technical report, Technical Report TR09-08, University of Alberta, Edmonton, 2009.

[Enz03] M. Enzenberger. "Evaluation in Go by a neural network using soft segmentation." In *Advances in computer games: many games, many challenges: proceedings of the ICGA/IFIP SG16 10th Advances in Computer Games Conference (ACG 10), November 24-27, 2003, Graz, Styria, Austria*, p. 97. Kluwer Academic Pub, 2003.

[FDM08] Dario Floreano, Peter Dürr, and Claudio Mattiussi. "Neuroevolution: from architectures to learning." *Evolutionary Intelligence*, **1**(1):47–62, March 2008.

[Fed04] Diego Federici. "Evolving a Neurocontroller Through a Process of Embryogeny." In Stefan Schaal, Auke Jan Ijspeert, Aude Billard, Sethu Vijayakumar, John Hal-

lam, and Jean-Arcady, editors, *Proceedings of the Eighth International Conference on Simulation and Adaptive Behavior (SAB-2004)*, pp. 373–384, Cambridge, MA, 2004. MIT Press.

[FFP90] D.B. Fogel, L.J. Fogel, and V.W. Porto. "Evolving neural networks." *Biological Cybernetics*, **63**(6):487–493, 1990.

[Fie02] Martin Fierz. "Simplech." http://arton.cunst.net/xcheckers/, 2002.

[Fie08] Martin Fierz. "Cake Checkers Engine." http://www.fierz.ch/cake.php, 2008.

[FM96] Dario Floreano and F. Mondada. "Evolution of Plastic Neurocontrollers for Situated Agents." **26**(3):396–407, 1996.

[Fog93] DB Fogel. "Using evolutionary programing to create neural networks that are-capable of playing tic-tac-toe." In *IEEE International Conference on Neural Networks, 1993.*, pp. 875–880, 1993.

[Fog02] David B. Fogel. *Blondie24: Playing at the Edge of AI.* 2002.

[Fra92] W N Martin Frank Z. Brill, D E Brown. "Fast generic selection of features for neural network classifiers." *IEEE Transactions on Neural Networks*, **3**(2):324–328, September 1992.

[FU00] Dario Floreano and Joseba Urzelai. "Evolutionary robots with online self-organization and behavioral fitness." *Neural Networks*, **13**:431–443, 2000.

[FU01] D. Floreano and J. Urzelai. "Evolution of Plastic Control Networks." *Autonomous Robots*, **11**(3):311–317, 2001.

[GC02] Geoffrey J. Goodhill and Miguel A. Carreira-Perpinn. "Cortical Columns." In L. Nadel, editor, *Encyclopedia of Cognitive Science*, volume 1, pp. 845–851. MacMillan Publishers Ltd., London, 2002.

[GFS07] A. Graves, S. Fernandez, and J. Schmidhuber. "Multi-dimensional recurrent neural networks." *Lecture Notes in Computer Science*, **4668**:549, 2007.

[GM97] Faustino Gomez and Risto Miikkulainen. "Incremental Evolution of Complex General Behavior." *Adaptive Behavior*, **5**:317–342, 1997.

[GR87] David E. Goldberg and J. Richardson. "Genetic Algorithms with Sharing for Multimodal Function Optimization." pp. 148–154, 1987.

[Gru95] Frederic Gruau. "Automatic Definition of Modular Neural Networks." *Adaptive Behaviour*, **3**(2):151–183, 1995.

[GS07a] Jason Gauci and Kenneth O. Stanley. "Generating Large-Scale Neural Networks Through Discovering Geometric Regularities." In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2007)*, New York, NY, 2007. ACM Press.

[GS07b] S. Gelly and D. Silver. "Combining online and offline learning in UCT." In *17th International Conference on Machine Learning*, pp. 273–280, 2007.

[GS07c] Sylvain Gelly and David Silver. "Combining Online and Offline Knowledge in UCT." In *In Zoubin Ghahramani, editor, Proceedings of the International Conference of Machine Learning (ICML 2007)*, pp. 273–280, 2007.

[GS08a] Jason Gauci and Kenneth O. Stanley. "A Case Study on the Critical Role of Geometric Regularity in Machine Learning." In *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence (AAAI-2008)*, Menlo Park, CA, 2008. AAAI Press.

[GS08b] Sylvain Gelly and David Silver. "Achieving Master Level Play in 9 x 9 Computer Go." In Dieter Fox and Carla P. Gomes, editors, *AAAI*, pp. 1537–1540. AAAI Press, 2008.

[GS09] A. Graves and J. Schmidhuber. "Offline handwriting recognition with multidimensional recurrent neural networks." *Advances in Neural Information Processing Systems*, **21**, 2009.

[GS10a] Jason Gauci and Kenneth O. Stanley. "Autonomous Evolution of Topographic Regularities in Artificial Neural Networks." *Neural Computation*, **22**(7):1860–1898, 2010. To appear.

[GS10b] Jason Gauci and Kenneth O. Stanley. "Indirect Encoding of Neural Networks for Scalable Go." pp. 354–363, 2010.

[GU92] Z. Guo and R. E. Uhrig. "Using Genetic Algorithms to Select Inputs for Neural Networks." *Proc. Int. Workshop Combinations of Genetic Algorithms and Neural Networks (COGANN-92)*, pp. 223–234, 1992.

[GWP96] Frederic Gruau, Darrell Whitley, and Larry Pyeatt. "A Comparison Between Cellular Encoding and Direct Encoding for Genetic Neural Networks." In John R. Koza, David E. Goldberg, David B. Fogel, and Rick L. Riolo, editors, *Genetic Programming 1996: Proceedings of the First Annual Conference*, pp. 81–89. MIT Press, 1996.

[HDO02] M.A. Hearst, ST Dumais, E. Osman, J. Platt, and B. Scholkopf. "Support vector machines." *Intelligent Systems and their Applications, IEEE*, **13**(4):18–28, 2002.

[HKB94] William E. Hart, Thomas E. Kammeyer, and Richard K. Belew. "The Role of Development in Genetic Algorithms." Technical Report CS94-394, University of California San Diego, San Diego, 1994.

[HL01] E. Hjelmas and B. K. Low. "Face Detection: A Survey." *Computer Vision and Image Understanding*, pp. 236–274, September 2001.

[HO01] N. Hansen and A. Ostermeier. "Completely derandomized self-adaptation in evolution strategies." *Evolutionary computation*, **9**(2):159–195, 2001.

[HP02] Gregory S. Hornby and Jordan B. Pollack. "Creating High-Level Components with a Generative Representation for Body-Brain Evolution." *Artificial Life*, **8**(3), 2002.

[Hub88] David H. Hubel. *Eye, Brain, and Vision (Scientific American Library).* W H Freeman & Co (Sd), 1988.

[HWL77] David H. Hubel, Torsten N. Wiesel, and S. LeVay. "Plasticity of Ocular Dominance Columns in Monkey Striate Cortex." **278**:377–409, 1977.

[IJC89] David J. Montana and Lawrence Davis. "Training Feedforward Neural Networks Using Genetic Algorithms." pp. 762–767, 1989.

[Kit90] Hiroaki Kitano. "Designing Neural Networks Using Genetic Algorithms with Graph Generation System." *Complex Systems*, **4**:461–476, 1990.

[KM75] D.E. Knuth and R.W. Moore. "An analysis of alpha-beta pruning." *Artificial Intelligence*, **6**(4):293–326, 1975.

[Koh81] Teuvo Kohonen. "Automatic Formation of Topological Maps of Patterns in a Self-Organizing System." In *Proceedings of the 2nd Scandinavian Conference on Image Analysis*, pp. 214–220, Espoo, Finland, 1981. Pattern Recognition Society of Finland.

[KR91] John R. Koza and James P. Rice. "Genetic generation of both the weights and architecture for a neural network." In *In International Joint Conference on Neural Networks*, pp. 397–404. IEEE, 1991.

[KS06] Levente Kocsis and Csaba Szepesvári. "Bandit based Monte-Carlo Planning." In *In: ECML-06. Number 4212 in LNCS*, pp. 282–293. Springer, 2006.

[KSJ91] Eric R. Kandel, James H. Schwartz, and Thomas M. Jessell, editors. *Principles of Neural Science*. Third edition, 1991.

[KSJ00] Eric R. Kandel, James H. Schwartz, and Thomas M. Jessell. *Principles of Neural Science*. McGraw-Hill, New York, fourth edition, 2000.

[LBH98] Yann Lecun, Yoshua Bengio, and Patrick Haffner. "Gradient-based learning applied to document recognition." In *Proceedings of the IEEE*, pp. 2278–2324, 1998.

[Lin74] A. Lindenmayer. "Adding Continuous Components to L-systems." In G. Rozenberg and A. Salomaa, editors, *L Systems, Lecture Notes in Computer Science 15*, pp. 53–68. Springer-Verlag, Heidelberg, Germany, 1974.

[LK94] G Leuba and R Kraftsik. "Changes in volume, surface estimate, three-dimensional shape and total number of neurons of the human primary visual cortex from midgestation until old age." volume 190, pp. 351–366. Springer Berlin / Heidelberg, 1994.

[LLE07] Bethany R. Leffler, Michael L. Littman, and Timothy Edmunds. "Efficient Reinforcement Learning with Relocatable Action Models." In *AAAI*, pp. 572–577. AAAI Press, 2007.

[Low04] D.G. Lowe. "Distinctive image features from scale-invariant keypoints." *International journal of computer vision*, **60**(2):91–110, 2004.

[LP00] Hod Lipson and Jordan B. Pollack. "Automatic Design and Manufacture of Robotic Lifeforms." *Nature*, **406**:974–978, 2000.

[LS04] J.M. Barreto L.O. Marin L.S. Encinas, A.C. Zimmermann. "Applying Dimensionality Reduction for Neural Networks Learning in the SORFACE Project." In *Artificial Intelligence and Applications*, volume 1. ACTA Press, 2004.

[Lu93] Chien-Ping Paul Lu. *Parallel search of narrow game trees.* PhD thesis, 1993.

[M01] Martin Müller. "Review: Computer Go 1984-2000.", 2001.

[Mar99] Andrew P. Martin. "Increasing Genomic Complexity by Gene Duplication and the Origin of Vertebrates." *The American Naturalist*, **154**(2):111–128, 1999.

[Mil04] Julian F. Miller. "Evolving a Self-Repairing, Self-Regulating, French Flag Organism." In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2004)*, Berlin, 2004. Springer Verlag.

[MM96] David E. Moriarty and Risto Miikkulainen. "Efficient Reinforcement Learning Through Symbiotic Evolution." In Leslie Pack Kaelbling, editor, *Recent Advances in Reinforcement Learning*. 1996.

[MRH86] James L. McClelland, David E. Rumelhart, and Geoffrey E. Hinton. "The Appeal of Parallel Distributed Processing." pp. 3–44. 1986.

[MSR91] Eric Mjolsness, David H. Sharp, and John Reinitz. "A Connectionist Model of Development." *Journal of Theoretical Biology*, **152**:429–453, 1991.

[Mul09] M. M

"uller. "Fuego at the Computer Olympiad in Pamplona 2009: a Tournament Report." Technical report, Technical Report TR09-00, University of Alberta, Edmonton, 2009.

[NYG98] A. Nakamura, T. Yamada, A. Goto, T. Kato, K. Ito, Y. Abe, T. Kachi, and R. Kakigi. "Somatosensory homunculus as drawn by MEG." *Neuroimage*, **7**(4):377–386, 1998.

[OS97] David W. Opitz and Jude W. Shavlik. "Connectionist Theory Refinement: Genetically Searching the Space of Network Topologies." **6**:177–209, 1997.

[kstanley: Uses Crossover, Direct Encoding, Topology manipulation TWGANN.]

[PP98] Joao Carlos Figueira Pujol and Riccardo Poli. "Evolving the Topology and the Weights of Neural Networks Using a Dual Representation." *Applied Intelligence Journal*, **8**(1):73–84, January 1998. Special Issue on Evolutionary Learning.

[RNC95] S.J. Russell, P. Norvig, J.F. Canny, J. Malik, and D.D. Edwards. *Artificial intelligence: a modern approach.* Prentice hall Englewood Cliffs, NJ, 1995.

[RS10] Sebastian Risi and Kenneth O. Stanley. "Indirectly Encoding Neural Plasticity as a Pattern of Local Rules." In *Proceedings of the 11th International Conference on Simulation of Adaptive Behavior (SAB2010)*, Berlin, 2010. Springer. To appear.

[RVH09] Sebastian Risi, Sandy D. Vanderbleek, Charles E. Hughes, and Kenneth O. Stanley. "How Novelty Search Escapes the Deceptive Trap of Learning to Learn." In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2009)*, New York, NY, 2009. ACM Press.

[SB98] R.S. Sutton and A.G. Barto. *Reinforcement Learning: An Introduction.* MIT Press, Cambridge, MA, 1998.

[SBM05] Kenneth O. Stanley, Bobby D. Bryant, and Risto Miikkulainen. "Evolving Neural Network Agents in the NERO Video Game." In *Proceedings of the IEEE 2005 Symposium on Computational Intelligence and Games*, 2005.

[SBM08a] Andrea Soltoggio, A. J. Bullinaria, Claudio Mattiussi, P. Dürr, and D. Floreano. "Evolutionary Advantages of Neuromodulated Plasticity in Dynamic, Reward-

based Scenarios." In Seth Bullock, Jason Noble, Richard Watson, and Mark Bedau, editors, *Proceedings of the Eleventh International Conference on Artificial Life (Alife XI)*, Cambridge, MA, 2008. MIT Press.

[SBM08b] Andrea Soltoggio, John A. Bullinaria, Claudio Mattiussi, Peter DÃŒrr, and Dario Floreano. "Evolutionary Advantages of Neuromodulated Plasticity in Dynamic, Reward-based Scenarios." In *Artificial Life XI*, pp. 569–576, Cambridge, MA, 2008. MIT Press.

[SDG09] Kenneth O. Stanley, David B. D'Ambrosio, and Jason Gauci. "A Hypercube-Based Indirect Encoding for Evolving Large-Scale Neural Networks." *Artificial Life*, **15**(2):185–212, 2009.

[SDS94] N.N. Schraudolph, P. Dayan, and T.J. Sejnowski. "Temporal difference learning of position evaluation in the game of Go." *Advances in Neural Information Processing Systems*, pp. 817–817, 1994.

[Sho03] Peter Shotwell. *Go! More Than a Game.* Turtle Publishing, 2003.

[Sim94] Karl Sims. "Evolving 3D Morphology and Behavior by Competition." pp. 28–39. MIT Press, Cambridge, MA, 1994.

[SM02] Kenneth O. Stanley and Risto Miikkulainen. "Evolving Neural Networks Through Augmenting Topologies." *Evolutionary Computation*, **10**:99–127, 2002.

[SM03] Kenneth O. Stanley and Risto Miikkulainen. "A Taxonomy for Artificial Embryogeny." *Artificial Life*, **9**(2):93–130, 2003.

[SM04a] Kenneth O. Stanley and Risto Miikkulainen. "Competitive Coevolution Through Evolutionary Complexification." **21**:63–100, 2004.

[SM04b] Kenneth O. Stanley and Risto Miikkulainen. "Evolving a Roving Eye for Go." In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2004)*, Berlin, 2004. Springer Verlag.

[SP91] M. Srinivas and L.M. Paranaik. "Learning neural network weights using genetic algorithms-improving performance by search-space reduction." *IEEE International Joint Conference on Neural Networks*, **3**:2331–2336, 1991.

[Spo02] Olaf Sporns. "Network Analysis, Complexity, and Brain Function." *Complexity*, **8**(1):56–60, 2002.

[SS05] Alexander A. Sherstov and Peter Stone. "Function Approximation via Tile Coding: Automating Parameter Choice." In Jean-Daniel Zucker and Lorenza Saitta, editors, *SARA*, volume 3607 of *Lecture Notes in Computer Science*, pp. 194–205. Springer, 2005.

[SS08] T. Schaul and J. Schmidhuber. "Scalable neural networks for board games." In *Proceedings of the International Conference on Artificial Neural Networks (ICANN)*. Springer, 2008.

[SSM07] D. Silver, R. Sutton, and M. Müller. "Reinforcement learning of local shape in the game of Go." In *20th International Joint Conference on Artificial Intelligence*, pp. 1053–1058, 2007.

[SSM08] D. Silver, R.S. Sutton, and M. Müller. "Sample-based learning and search with permanent and transient memories." In *Proceedings of the 25th international conference on Machine learning*, pp. 968–975. ACM, 2008.

[Sta03] Kenneth O. Stanley. *Efficient Evolution of Neural Networks Through Complexification.* PhD thesis, 2003.

[Sta06a] Kenneth O. Stanley. "Comparing Artificial Phenotypes with Natural Biological Patterns." In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO) Workshop Program*, New York, NY, 2006. ACM Press.

[Sta06b] Kenneth O. Stanley. "Exploiting Regularity Without Development." In *Proceedings of the AAAI Fall Symposium on Developmental Systems*, Menlo Park, CA, 2006. AAAI Press.

[Sta07] Kenneth O. Stanley. "Compositional Pattern Producing Networks: A Novel Abstraction of Development." *Genetic Programming and Evolvable Machines Special Issue on Developmental Systems*, **8**(2):131–162, 2007.

[Swi96] N. V. Swindale. "The Development of Topography in the Visual Cortex: A Review of Models." **7**:161–247, 1996.

[TL05] Julian Togelius and Simon M. Lucas. "Forcing Neurocontrollers to Exploit Sensory Symmetry Through Hard-wired Modularity in the Game of Cellz." In *Proceedings of the IEEE Symposium on Computational Intelligence and Games*, pp. 37–43, 2005.

[Tur52] Alan Turing. "The Chemical Basis of Morphogenesis." *Philosophical Transactions of the Royal Society B*, **237**:37–72, 1952.

[VS10a] Phillip Verbancsics and Kenneth O. Stanley. "Evolving Static Representations for Task Transfer." *Journal of Machine Learning Research (JMLR)*, 2010. To appear.

[VS10b] Phillip Verbancsics and Kenneth O. Stanley. "Task Transfer through Indirect Encoding." In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2010)*, New York, NY, 2010. ACM Press.

[WHR87] James D. Watson, Nancy H. Hopkins, Jeffrey W. Roberts, Joan A. Steitz, and Alan M. Weiner. *Molecular Biology of the Gene Fourth Edition*. The Benjamin Cummings Publishing Company, Inc., Menlo Park, CA, 1987.

[WKM05] Shimon Whiteson, Nate Kohl, Risto Miikkulainen, and Peter Stone. "Evolving Keepaway Soccer Players Through Task Decomposition." *Machine Learning*, **59**:5–30, 2005.

[Yao99] Xin Yao. "Evolving Artificial Neural Networks." *Proceedings of the IEEE*, **87**(9):1423–1447, 1999.

[YGS93] TT Yang, CC Gallen, BJ Schwartz, and FE Bloom. "Noninvasive somatosensory homunculus mapping in humans by using a large-array biomagnetometer." *Proceedings of the National Academy of Sciences*, **90**(7):3098–3102, 1993.

[ZBL99] Michael J. Zigmond, Floyd E. Bloom, Story C. Landis, James L. Roberts, and Larry R. Squire, editors. *Fundamental Neuroscience*. Academic Press, London, 1999.

[ZM93] Byoung-Tak Zhang and Heinz Muhlenbein. "Evolving Optimal Neural Networks Using Genetic Algorithms with Occam's Razor." **7**:199–220, 1993.

[kstanley: Uses tree representation and subtree crossover in TWGANN.]